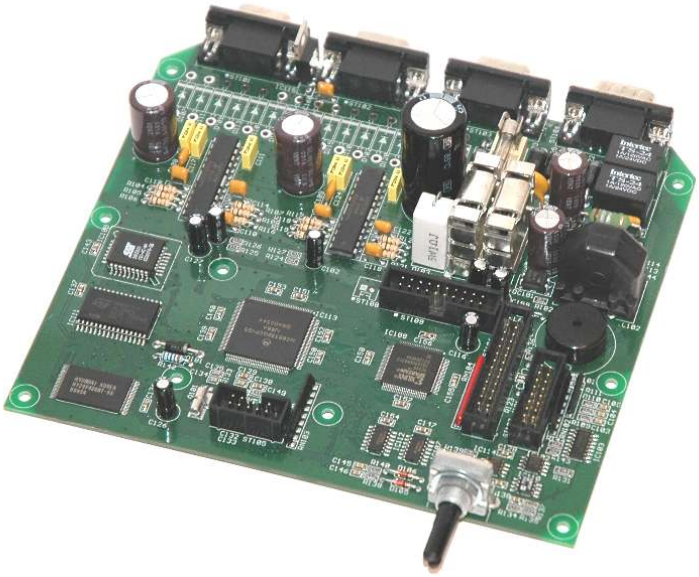
Motorsteuerungen CO-Serie

Benutzerhandbuch

Version 1.28



Calantec GmbH Automatisierungstechnik, Industrieelektronik

Nordufer 4
13353 Berlin
http://www.calantec.de

Telefon: 030 453 01 519
Telefax: 030 453 01 531
email: info@calantec.de

Geschäftsführer: Hartmut Schäfer
Handelsregister: Amtsgericht Charlottenburg HRB 92065
USt-ID: DE232787423

Gültig für Steuerungen ab Software-Version 016.031
Baureihen CO2200, CO4200, CO4300, CO6100, CO6150,
CO6300, CO6500, CO5400, CO5500

H. Schäfer 18. November 2004

Inhaltsverzeichnis

1. Anschluß und Inbetriebnahme 4
Konfiguration der Achsen 4
Einstellung der PID-Parameter 4
Organisation der Achsen 5
CONFIG.SYS-Datei 6
Eigenschaften der FLASH und EEPROM-Disk 6
BASIC-Interpreter 7
Beispielprogramm 7
Update des Betriebssystems 8
Verlängerung des Schnittstellenkabels 9
Anschluß von Endstufen mit analogem Eingang 9
Programmierung mit SOSCom 9

2. Befehlsreferenz 11
BASIC-Anweisungen und Funktionen 11
Motorsteuerungs-Anweisungen 48

3. Anhang A: G-Codes 71
Standard-G-Codes 71
G-Code Ergänzungen für spezielle Programme: 81

4. Anhang B: Programmierhinweise für spezielle Hardware 83
Programmierung der Mikroschrittendstufen: 83
Hinweise zu den Mikroschrittendstufen der Steuerung CO6300 84
Verwendung eines Digitalpotentiometers als Eingabehilfe: 84

5. Anhang C: Ein- und Ausgänge, Übersicht und Beschaltung 86
Eingänge 86
Ausgänge 88
Beschaltung (Ein- und Ausgänge) 90
Beschaltung NOTAUS-Funktionen 91
SPS-Anschluß 92

6. Anhang D: ASCII-Code Tabelle 94

7. Anhang E: Anschlußpläne 96

8. Anhang F: Beschreibung der Anschlüsse 98

1. Anschluß und Inbetriebnahme

Die Versorgungsspannung der Steuerungen sollte in der Regel 24V+/-10% betragen (kundenspezifische Steuerungen können abweichende Versorgungsspannungen haben). Je nach Modell und angeschlossener Peripherie (Inkrementalgeber, Endstufen, Ausgänge) liegt die Stromaufnahme bei ca. 0.5 A bis 2 A (ohne Endstufenversorgung).
Um die Steuerung zu programmieren muß sie mit einem Terminal oder einem Computer mit serieller Schnittstelle und einem Terminalprogramm verbunden werden. Die Standard-Einstellungen sind: Baudrate 19200 Baud, keine Parität, 8 Datenbits, 1 Stopbit, kein Handshake. Diese Einstellungen können mit dem `MODE`-Befehl (siehe Befehlsreferenz) verändert werden. Falls die Steuerung mit einer RS232-Schnittstelle bestückt ist, kann die Verbindung zum Computer mit einem Standard-Nullmodemkabel erfolgen. Hat die Steuerung eine RS422-Schnittstelle, ist ein RS422/RS232-Konverter einzusetzen.

Konfiguration der Achsen

Nach dem Einschalten der Steuerung sollten zuerst die Achsen konfiguriert werden. Dies geschieht am besten in der Datei `CONFIG.SYS`. Diese Datei wird nach dem Einschalten automatisch ausgeführt, falls sie vorhanden ist. Zur Konfiguration wird die `MODE` und die `SET`-Anweisung verwendet. Für jede der Achsen wird festgelegt, ob es sich um eine Servomotor- oder eine Schrittmotorachse handelt, und welche Art von Endstufe angeschlossen ist. Für die PWM2100-Endstufen muß die Option `PWMS` gewählt werden, für die Endstufen SM1100 und SM2100 die Option `STEPDIR`. Für kundenspezifische Steuerungen mit integrierten Endstufen sind die vorgegebenen Parameter zu verwenden.

Beispielkonfiguration:

```
MODE 1,PID,INC(1),PWMS(1)
MODE 2,PID,INC(2),PWMS(2)
MODE 3,SM,STEPDIR(3)
```

Achse 1 und Achse 2 werden zum Anschluß von Servomotoren konfiguriert, Achse 3 als Schrittmotor. Danach sollten mit der `SET`-Anweisung weitere Parameter eingestellt werden, insbesondere der maximale Dauerstrom (`IMAX`, bei Schrittmotoren auch der Ruhestrom `ISTANDBY`) und der Spitzenstrom `IPEAK`. Für den PWM-Ausgang sind sie Parameter `PWMPOL` und `PWMOFFSET` anzugeben. Bei Servomotoren sind die PID-Parameter (`POL`, `DIFF`, `PROP`, `INT`, `DEADBAND`) einzustellen. Zum Einstellen von `POL` und zur Programmierung der Referenzfahrten ist empfehlenswert, den Regelausgang mit `MAXFORCE` zu begrenzen. Mikroschritt-endstufen müssen je nach Ausführung gegebenenfalls mit den gewünschten Phasenstromfunktionen programmiert werden (`WAVEFORM1`, `WAVEFORM2`)

Einstellung der PID-Parameter

Die Einstellung der Parameter für den PID-Regler muß sorgfältig geschehen, um die höchstmögliche Geschwindigkeit und Genauigkeit zu erreichen. Dazu erstellt man am besten eine Datei, in der die Anweisungen zur Achskonfiguration aufgeführt sind:

```
MODE 1,PID,INC(1),PWMS(1)
SET 1,IMAX,10
```

```
SET 1,IPEAK,10
SET 1,PWMPOL,TRUE
SET 1,POL,TRUE
SET 1,MAXFORCE,10
SET 1,PROP,100
SET 1,INT,0
SET 1,DIFF,200
```

Für weitere Achsen können die entsprechenden Anweisungen angefügt werden. Man beginnt zunächst mit einem, stark begrenzten Reglerausgang (`SET 1,MAXFORCE,10` oder weniger) und, sofern diese das unterstützen, mit begrenztem Endstufenstrom (`SET 1,MAXFORCE,10`). Nun schaltet man den Regler mit `MOTOR 1 ON` ein. Da die Regelabweichung zunächst Null ist, passiert zunächst nichts. Wird nun die Motorachse bewegt, wird der Motor entweder zurückregeln, wie es gewünscht ist, oder in eine Richtung weiterlaufen, ohne anzuhalten. Dann ist die Regelpolarität falsch eingestellt, d.h. aus `SET 1,POL,TRUE` muß `SET 1,POL,FALSE` oder umgekehrt werden. Die Polaritätsumkehr kann auch durch Vertauschen der Motoranschlüsse oder Vertauschen von Kanal A und B der Inkrementalgebereingänge erreicht werden. Stimmt die Regelpolarität, können die Werte der Parameter `MAXFORCE`, `IMAX` und `IPEAK` etwas erhöht werden. Ausgehend von den oben gezeigten Werten müssen nun möglichst gute Werte für den Proportional- und den Differentialanteil des Reglers gefunden werden. Dabei erhöht man zunächst schrittweise den Proportionalanteil, bis der Regler anfängt zu schwingen.
Häufig muß das Schwingen provoziert werden, entweder durch Anregung von Hand oder durch die `JUMP` Anweisung. Den Proportionalanteil stellt man so ein, daß der Regler gerade nicht mehr oder nur sehr schwach schwingt. Dann erhöht man den Differentialanteil, bis der Regler auch bei stärkerer Auslenkung sicher wieder in die Sollposition einregelt, dabei sollte er nur minimal überschwingen. Wenn das Regelverhalten den Anforderungen entspricht, können die Werte der Parameter `MAXFORCE`, `IMAX` und `ISTANDBY` auf den gewünschten Endwert gesetzt werden.
Es kommt häufig, inbesondere beim Anschluß von Endstufen mit analogem Eingang zu Geräuschen beim Stillstand des Motors, die durch minimale Reglerschwingungen um 1 oder 2 Inkremente verursacht werden. Je nach Endstufe und Mechanik können diese entweder durch Verringerung des Proportionalanteils, durch Einstellung eines Offsets (siehe `SET na,OFFSET`), oder durch ein Regler-Totband (siehe `SET na,DEADBAND`) reduziert oder beseitigt werden. Ursache kann auch eine zu indirekte Kopplung von Motor und Inkrementalgeber sein, falls der Inkrementalgeber z.B. nicht direkt auf der Motorachse montiert ist. Dies wirkt sich ungünstig auf das Regelverhalten aus und sollte vermieden werden.

Organisation der Achsen

Sind die Achsen konfiguriert, werden sie mit der `GROUP`-Anweisung zu Gruppen zusammengefaßt, innerhalb derer interpoliert werden kann. Alle Anweisungen zur Bewegungssteuerung setzen eine Gruppe voraus, die aus 1, 2 oder 3 Achsen bestehen kann.

Beispiel:

```
GROUP 1,1,2,3
```

Gruppe 1 besteht damit aus den Achsen 1, 2 und 3. Die Reihenfolge bei der

Gruppendefinition gibt an, welche Achse danach als X-, Y- oder Z-Achsen verwendet werden. Mit der TRAFO-Anweisung können dann die Maschinenkoordinaten (bzw. die sogenannten Quadcounts der Inkrementalgeber) in benutzerdefinierte Koordinaten (z.B. mm) umgerechnet werden. Es können auch Winkelfehler korrigiert werden.

CONFIG.SYS-Datei

Die CO-Baureihen besitzen je ein FLASH- und ein EEPROM-Laufwerk. Das FLASH-Laufwerk hat den Namen SID0, das EEPROM-Laufwerk den Namen SID1. Falls bei Dateioperationen der Laufwerksnamen weggelassen wird, wird SID1 verwendet. SID0 sollte nur für selten zu ändernde Dateien (CONFIG.SYS, Programmdateien) verwendet werden. Änderungen an SID0 werden zunächst nur in einer gespiegelten RAM-Kopie vorgenommen, die Aktualisierung erfolgt erst nach der Anweisung UPDATE "SID0". Während der Aktualisierung ist die Motorregelung unterbrochen.
Beim Einschalten wird automatisch die Datei CONFIG.SYS ausgeführt (wie mit EXEC), bei der CO-Baureihe von Laufwerk SID0. Dies kann verhindert werden, indem während des Einschaltens die **ESC**-Taste gedrückt wird (bei kundenspezifischen Steuerungen kann dies auch ein Schalter oder Taster sein).
Die CONFIG.SYS-Datei wird am einfachsten erzeugt, indem der Editor mit EDIT "CONFIG.SYS" bzw. EDIT "SID0:CONFIG.SYS" aufgerufen wird und die Anweisungen dann entweder von Hand eingegeben oder mittles ASCII-Transfer des Terminalprogramms heruntergeladen werden. Die Datei kann dann mit **MENU/SAVE** abgespeichert werden. Bei der Ausführung werden Befehle, denen keine Zeilennummer vorangeht, sofort ausgeführt, Programmzeilen werden in den Programmspeicher geladen, jedoch nicht ausgeführt. Eine Ausführung innerhalb der CONFIG.SYS-Datei (bzw. innerhalb jeder mit EXEC aufgerufenen Datei) erfolgt mit der RUN-Anweisung.

Beispiel:
```
NOTAUS INTERN
MODE 1,SM,PATTERN(1)
MODE 2,SM,PATTERN(2)
SET 1,IMAX,30
SET 2,IMAX,30
SET 1,ISTANDBY,20
SET 2,ISTANDBY,20
GROUP 1,1,2
TRAFO 1,534,0,0,534
EXEC "PROGRAMM.SYS"
RUN
```

Eigenschaften der FLASH und EEPROM-Disk

Die eingebaute EEPROM-Disk (SID1:) hat eine Kapazität von 256 K-Byte. Es können maximal 128 Dateien angelegt werden. Die Formatierung geschieht mit der FORMAT-Anweisung. Das Inhaltsverzeichnis kann mit FILES abgerufen werden, dabei ist zu beachten, daß Dateien, die die Erweiterung .SYS haben, nicht angezeigt werden. Die Anzeige kann mit FILES "*.*",1 erzwungen werden.
Die FLASH-Disk SID0: hat ebenfalls eine Kapazität von 256 K-Bytes. Zur Bearbeitung wird eine Kopie der Disk im RAM abgelegt, daher wird mindestens

256 K-Bytes nichtfragmentierter Arbeitsspeicher benötigt. An dieser RAM-Kopie werden dann alle Änderungen vorgenommen wie z.B. formatieren oder speichern und löschen von Dateien. Mit der UPDATE Anweisung wird die geänderte Kopie dann wieder in den FLASH-Speicher kopiert.

BASIC-Interpreter

Nach dem Einschalten meldet sich die Steuerung, falls keine CONFIG.SYS-Datei angelegt wurde, die dies verhindern könnte, über die Schnittstelle STD: (Tastatur und COM1: beim Lesen, Display und COM1: beim Schreiben) mit der Eingabeaufforderung > (Prompt). Nun können Anweisungen eingegeben werden, die sofort nach dem Zeilenende (CR oder CR/LF) ausgeführt werden. Die über STD: eingehenden Zeichen werden an STD: wieder zurückgesandt (Echo). Die Eingabezeile kann mit Backspace korrigiert werden. Diese Eigenschaften können mit den Anweisungen PROMPT, ECHO und SHELL verändert werden.
Ein Programm wird erstellt, indem Anweisungszeilen mit Zeilennummern eingegeben werden. Das aktuelle Programm kann mit LIST überprüft werden. Das im Speicher befindliche Programm kann mit der SAVE-Anweisung gespeichert und mit LOAD wieder geladen werden.
Es ist empfehlenswert, die Grundkonfiguration (Achskonfiguration, Baudrate der Schnittstelle, Achstransformation etc.) in der CONFIG.SYS-Datei zu speichern und das Ablaufprogramm in einer weiteren Datei abzulegen, welche innerhalb der CONFIG.SYS aufgerufen und gestartet wird. Umfangreiche Programme sollten mit einem Texteditor verfaßt und dann über die serielle Schnittstelle in die Steuerung übertragen werden (vorher den Speicher mit NEW löschen). Dazu kann die ASCII-Transferfunktion des Terminalprogramms verwendet werden. Das Programm kann dann mit SAVE abgespeichert werden. Dateien, welche Befehle ohne Zeilennummern enthalten (z.B. die CONFIG.SYS-Datei), können übertragen werden, indem zunächst der Editor mit EDIT aufgerufen, die Datei übertragen und mit **MENU/SAVE** abgespeichert wird.

Beispielprogramm

Ein einfaches Beispielprogramm kann auf folgende Weise programmiert werden: Nach dem Einschalten der Steuerung gibt man, sofern die Eingabeaufforderung erscheint, folgende Zeilen ein:

```
>
>100 FOR i=1 to 100
>110   PRINT "HALLO"
>120   SLEEP 200
>130 NEXT
>
```

Falls die Eingabeaufforderung nicht erscheint, sind die Einstellungen des Terminal-Programmes und die Kabelverbindung zu überprüfen. Gegebenenfalls muß die Steuerung mit gedrückter **ESC**-Taste eingeschaltet werden, damit eine eventuell vorhandene CONFIG.SYS-Datei nicht ausgeführt wird.
Das oben eingegeben Programm kann mit der Anweisung LIST überprüft werden. Mit der Anweisung RUN wird das Programm gestartet. Im 200 ms Abstand wird dann der 100 mal der Text HALLO ausgegeben. Unterbrochen wird die Ausführung mit der

Tastenkombination Strg-C (ASCII-Code 3). Achtung, einige Terminalprogramme akzeptieren diese Eingabe nicht. Eventuell kann dies über Funktionstasten emuliert werden.
Das Programm kann mit der Anweisung

```
>SAVE "TEST.BAS"
```

abgespeichert werden. Ohne Angabe des Gerätenamens wird standardmäßig die EEPROM-Disk `SID1:` verwendet. Das Inhaltsverzeichnis von `SID1:` erhält man mit der Anweisung

```
>FILES "SID1:"
```

oder einfach nur

```
>FILES
```

Soll dieses Programm beim Einschalten automatisch aufgerufen werden, ist eine entsprechende `CONFIG.SYS`-Datei zu erstellen. Diese muß auf der FLASH-Disk `SID0:` abgelegt werden. Da die Befehle in der CONFIG.SYS direkt aufgerufen werden sollen, ist eine Programmierung über Zeilennummern und Abspeichern mit SAVE nicht möglich. Es muß der Editor verwendet werden.

```
>EDIT "SID0:CONFIG.SYS"
```

Nun kann der Text eingegeben werden. Für das Beispielprogramm genügt eine Anweisung zum Laden des Programmes und eine Anweisung zum Programmstart.

```
EXEC "TEST.BAS"
RUN
```

Längere CONFIG.SYS-Dateien können mit Hilfe der Text-Upload-Funktion des Terminalprogrammes in den Editor geladen werden. Die Datei muß dann mit **MENU/SAVE** abgespeichert werden. Danach kann der Editor mit **MENU/EXIT** verlassen werden. Im Gegensatz zur EEPROM-Disk `SID1:` muß die FLASH-Disk `SID0:` nun aktualisiert werden. Dies geschieht durch die Anweisung

```
>UPDATE "SID0:"
```

Die Ausführung dauert einige Sekunden. Technisch bedingt werden dabei alle Endstufenausgänge abgeschaltet. Beim nächsten Einschalten wird nun das Programm automatisch aufgerufen.
Das Inhaltsverzeichnis von `SID0:` erhält man mit der Anweisung

```
>FILES "SID0:",1
```

Der Zusatz `,1` bewirkt dabei, daß auch Dateien mit der Endung `.SYS` wie z.B. die `CONFIG.SYS` angezeigt werden.

Update des Betriebssystems

Das Betriebssystem der Steuerungen kann durch neuere Versionen ersetzt werden. Die aktuelle Version erhält man mit der Anweisungen

```
>PRINT VERSION$
```

Zum Update benötigt man ein Terminalprogramm und die zum Steuerungsmodell passende `UPDATE.DAT` Datei. Das Update wird mit der Anweisung

```
>UPDATE 12345
```

gestartet. Die Steuerung quittiert diese Anweisung zunächst mit der Ausgabe der Zahlen 1 und 2 (Initialisierung). Gleichzeitig werden bei Steuerungen mit großer Tastatur die **REF-** und die **TEACH**-LED eingeschaltet. Nun überträgt man die Datei `UPDATE.DAT` mit der Textdateiübertragungsfunktion des Terminalprogramms. Während der Übertragung blinkt die **EDIT**-LED langsam. Nach der Übertragung, die einige Minuten dauern kann, wird das Betriebssystem-Update durchgeführt. Dies wird durch Ausgabe der Zahlen 3,4 und 5 und Einschalten der **MOVE**- und **FUNC**-LED bestätigt. In der letzten Phase darf die Stromversorgung der Steuerung nicht unterbrochen werden.
Nach dem vollständigen Update führt die Steuerung einen `RESET` aus.

Verlängerung des Schnittstellenkabels

Das Kabel zu den RS422-Schnittstellen kann bedenkenlos auf einige -zig Meter verlängert werden. Beim Anschluß eines RS232/RS422 Interfaces sollte die RS422-Seite verlängert werden, nicht die RS232-Seite. Bei Baudraten bis 19200 Baud sollten auf der RS232-Seite 10 m nicht überschritten werden.
Die RS422-Verlängerung kann durch ein mindestens 10-adriges abgeschirmtes Kabel erfolgen, bei größeren Längen sollten beide Masse- und +5 V-Verbindungen mit einem Kabel verbunden werden, um die Verluste im Kabel zu reduzieren. Falls auf RTS/CTS-Handshake verzichtet werden kann, kann die Verlängerung auch mit einem 6-adrigen Kabel erfolgen. Die Anschlußbelegungen der Schnittstellen sind im Anhang zu finden.

Anschluß von Endstufen mit analogem Eingang

Der Anschluß von Endstufen mit analogem Eingang ist bei den Steuerungen CO2200 und CO4200 möglich. Diese besitzen einen analogen Ausgang (AOUTn), der +/- 10 V liefern kann. Bei der Achskonfiguration ist dieser anstelle des PWM-Ausgangs anzugeben, beispielsweise:

```
MODE 1,PID,INC(1),DAOUT(1)
```

Zusätzlich ist der Anschluß der EN-Leitung (Enable) an die Endstufe nötig, da die Steuerung beim Ausschalten undefinierte Spannungen am analogen Ausgang liefern kann. Die EN-Leitung liefert +5 V, wenn die Endstufe eingeschaltet werden soll, und 0 V, wenn sie ausgeschaltet werden soll. Der Pegel ist gegebenenfalls über Inverter oder Pegelwandler anzupassen.
Die analogen Ausgänge zur Stromeinstellung (DA-A, DA-B) sind dann, ebenso wie die entsprechenden Anweisungen (`SET na,IMAX,a`, `SET na,IPEAK,a`) funktionslos, falls sie nicht von den Endstufen unterstützt werden. Die Spannungspegel an den Ausgängen DA-A und DA-B sind 0..5 V, die Ausgänge dürfen nur hochohmig belastet werden.

Programmierung mit SOSCom

Zur Vereinfachung der Kommunikation und dem Einstellen der Regelparameter kann auch das Programm SOSCom verwendet werden. Es kann bei Calantec erhältlich.

2. Befehlsreferenz

Variablentypen:

`FIX`	64 Bit-Zahl mit 32 Vor- und 32 Hinterkommastellen
`INTEGER`	32 Bit Integerzahl
`STRING`	Zeichenkettenausdruck beliebiger Länge
`CHAR`	einzelnes 8 Bit Zeichen

BASIC-Anweisungen und Funktionen

Konventionen:

`a`	numerischer Ausdruck (Integer oder Fix)
`n`	numerischer Ausdruck (Integer)
`nb`	Boolscher Ausdruck
`c`	Einzelnes ASCII-codiertes Zeichen
`s$`	Zeichenkette (String)
`a(n)`	Array
`[..]`	optionaler Parameter

ABS *Funktion*

Liefert den Absolutwert eines numerischen Ausdrucks. Ein Fixkommawert ergibt ein Fixkommaergebnis, ein Integerwert ein Integerergebnis.

Syntax:

```
a=ABS(a)
```

Beispiel:

```
PRINT ABS(x)
PRINT ABS(-100.4)
```

Siehe auch:

`INT, FRAC`

ACOS *Funktion*

Liefert den Arcuscosinus eines numerischen Ausdrucks.

Syntax:

```
a=ACOS(a)
```

Beispiel:

```
PRINT ACOS(x)
PRINT ACOS(0.4)
```

Siehe auch:

`SIN, COS, ASIN`

AIN *Funktion*

Liefert den Wert eines analogen Signales am Eingang `n`. Dabei wird der Wert `100` bei der maximalen Eingangsspannung, der Wert `-100` bei der minimalen Eingangsspannung angenommen.

Syntax:

```
a=AIN(n)
```

Beispiel:

```
PRINT AIN(2)
```
Siehe auch:
```
CALIBRATE, AOUT
```

AOUT *Anweisung*

Gibt den angegebenen Wert `a` am analogen Ausgang `n` aus. Dabei ist `a=100` der Maximalwert, `a=-100` der Minimalwert, wenn es sich um einen bipolaren Ausgang (AOUT) handelt, `a=0`, wenn es sich um einen Unipolaren Ausgang handelt (DA-A, DA-B).

Syntax:
```
AOUT n,a
```
Beispiel:
```
AOUT 1,15
```
Siehe auch:
```
CALIBRATE, AIN
```

ASC *Funktion*

Liefert den ASCII-Code eines Zeichen bzw. des ersten Zeichens eines Strings.

Syntax:
```
n=ASC(c)
n=ASC(s$)
```
Beispiel:
```
PRINT ASC(”Hallo”)
```
Siehe auch:
```
CHR$
```

ASHIFT *Funktion*

Verschiebt eine 32-Bit-Zahl `n1` arithmetisch bitweise um `n2` Stellen.

Syntax:
```
n=ASHIFT(n1,n2)
```
Siehe auch:
```
ROTATE, SHIFT
```

ASIN *Funktion*

Liefert den Arcussinus eines numerischen Ausdrucks.

Syntax:
```
a=ASIN(a)
```
Beispiel:
```
PRINT ASIN(x)
PRINT ASIN(0.4)
```
Siehe auch:
```
SIN, COS, ACOS
```

ASSIGN *Anweisung*

Weist Hilfsgeräten (`STD:`, `AUX1..4:`) ein Ein- oder Ausgabegerät zu. Gerät `s1$` wird dem Hilfsgerät `s2$` als

Eingabegerät (Option `INPUT`) oder Ausgabegerät (Option `OUTPUT`) zugeordnet. Mit der Option `APPEND` wird `s1$` zusätzlich zu den bereits zugewiesenen Geräten verwendet, d.h. Ausgaben werden auf allen Geräten ausgegeben bzw. Eingaben von allen Geräten gleichermassen gelesen. Ohne `APPEND` werden bereits vorhandene Zuweisungen überschrieben.

Syntax:

```
ASSIGN s1$,s2$,INPUT|OUTPUT[,APPEND]
```

Beispiel:

```
ASSIGN "DIS:","AUX1:",OUTPUT
ASSIGN "KEY:","AUX1:",INPUT
ECHO "AUX1:",ON
```

Siehe auch:

```
OPEN, CLOSE, PRINT
```

BEEP *Anweisung*

Variante 1: Schaltet Tastatursignal ein oder aus.

Syntax:

```
BEEP ON|OFF
```

Variante 2: Der Signalton wird für die in ms angegebene Zeit `n` eingeschaltet.

Syntax:

```
BEEP n
```

Beispiel:

```
BEEP 500
```

BIN$ *Funktion*

Liefert die `n2`-stellige Binärdarstellung einer Integerzahl `n1` als String.

Syntax:

```
s$=BIN$(n1,n2)
```

Beispiel:

```
PRINT BIN$(123,8)
```

Siehe auch:

```
OCT$, HEX$
```

BREAK *Anweisung*

Schaltet die Unterbrechungsmöglichkeit mit Ctrl-C (ASCII-Code 3) bzw. der **STOP**-Taste ein oder aus.

Syntax:

```
BREAK ON|OFF
```

CALIBRATE *Anweisung*

Korrigiert den Nullpunkt des analogen Ausgangs `n1` um den Wert `n2`. Falls die Ausgänge nicht als Steuerausgänge für Servomotorendstufen verwendet werden, wird der Korrekturwert bei der nächsten `AOUT`-Anweisung berücksichtigt. Die Korrekturwerte können mit der Anweisung `WRITEPREFS` dauerhaft bespeichert werden.

Syntax:
```
CALIBRATE n1,n2
```
Beispiel:
```
CALIBRATE 1,-3
```
Siehe auch:
```
AOUT, WRITEPREFS
```

CAN

Anweisung (nur mit CAN Hardware)

Schaltet die CAN-Ereignisverfolgung für CAN-Kanal `n` (`n=1..4`) ein oder aus.

Syntax:
```
CAN(n) ON|OFF
```
Siehe auch:
```
ON xx GOTO|GOSUB
```

CANBAUD

Anweisung (nur mit CAN Hardware)

Setzt die Baudrate von CAN-Interface `n1` auf den Wert `n2`.

Syntax:
```
CANBAUD n1,n2
```
Beispiel:
```
CANBAUD 1,125000
```
Siehe auch:
```
CANID, CANERROR, CANREAD, CANWRITE
```

CANERROR

Funktion (nur mit CAN Hardware)

Gibt den Code des letzten Fehlers des CAN-Interface `n` zurück. Die Fehlercodierung ist noch vorläufig.

Syntax:
```
n=CANERROR(n)
```
Beispiel:
```
PRINT CANERROR(1)
```
Siehe auch:
```
CANBAUD, CANID, CANREAD, CANWRITE
```

CANID

Anweisung (nur mit CAN Hardware)

Setzt die bei der CAN-Kommunikation verwendeten Identifier. Kanal 0 wird für die serielle Kommunikation über den CAN-Bus verwendet. Diese findet zwischen einem Master und mehreren Slaves statt. Die Kommunikation vom Master zum Slave nutzt dabei einen Satz von 32 Identifiern (Basis `IDSource`, Identifier `IDSource` bis `IDSource+31`) die von den Slaves zum Master einen weiteren (Basis `IDTarget`, Identifier `IDTarget` bis `IDTarget+31`). `IDSource` und `IDTarget` müssen ganzzahlige Vielfache von 32 sein. Dabei sind für `IDSource` beim Master und für `IDTarget` bei den Slaves die gleichen Werte einzutragen, ebenso für `IDTarget` beim Master und `IDSource` bei den Slaves.
Bei den Kanälen 1 bis 4 wird der Identifier (`ID`) der CAN-Nachrichten angegeben, die von den Kanälen empfangen werden sollen. Bei Kanal 1 ist zusätzlich die Angabe einer Identifier-Maske möglich. Dabei werden beim Empfang die Bitstellen des Identifiers nicht verglichen, die in der Maske auf 0 gesetzt sind.
Der Parameter `mode` gibt an, ob das Standard- oder das Extended-Format für die CAN-Identifier verwendet wird. Ist `mode=0`, wird das Standard-Format mit 10 Bit Länge verwendet, ist `mode=1` das Extended-Format mit 29 Bit Länge.

Syntax:

```
CANID 0,mode,IDSource,IDTarget
CANID n,mode,ID[,mask]
```

Beispiel:

```
CANID 0,0,64,128
CANID 1,0,121,&hffffffff0
CANID 2,0,34
```

Siehe auch:

`CANBAUD, CANERROR, CANREAD, CANWRITE, OPEN`

CANREAD

Anweisung (nur mit CAN Hardware)

Liest eine CAN-Nachricht in ein Array. `n` gibt die Kanalnummer (1 bis 4) an. Das Array muß ein eindimensionales `array of char` sein und mindestens die Größe 16 haben. Die Arrayfelder werden wie bei der Anweisung `CANWRITE` verwendet, sie müssen nicht vorinitialisiert werden. Zusätzlich wird im Feld 16 der Wert 0 übergeben, falls keine neue Nachricht empfangen wurde, und der Wert `&hff`, falls eine neue Nachricht vorlag.

Syntax:

```
CANREAD n,array(0)
```

Beispiel:

```
DIM candata(16) AS CHAR
CANREAD 1,candata(0)
```

Siehe auch:

`CANBAUD, CANERROR, CANID, CANWRITE`

CANWRITE *Anweisung (nur mit CAN Hardware)*

Sendet eine Nachricht über den CAN-Bus Schnittstelle `n`. Das Array, in dem die zu sendende Nachricht übergeben wird muß ein eindimensionales `array of char` sein und mindestens die Größe 16 haben. Die Felder müssen wie folgt ausgefüllt werden:

`array(1)`	Identifier, höchstwertiges Byte
`array(2)`	Identifier, zweithöchstes Byte
`array(3)`	Identifier, dritthöchstes Byte
`array(4)`	Identifier, niedrigstwertiges Byte
`array(5)`	Datenbyte 0
`array(6)`	Datenbyte 1
`array(7)`	Datenbyte 2
`array(8)`	Datenbyte 3
`array(9)`	Datenbyte 4
`array(10)`	Datenbyte 5
`array(11)`	Datenbyte 6
`array(12)`	Datenbyte 7
`array(13)`	reserviert
`array(14)`	Anzahl der Datenbytes
`array(15)`	Modus
`array(16)`	reserviert

Der Wert für Modus ist entweder 0 für das Standard-Format des Identifiers (10 Bit) oder 1 für das Extended-Format des Identifiers (29 Bit).

Syntax:

```
CANWRITE n,array(0)
```

Beispiel:

```
DIM candata(16) AS CHAR
candata(1)=0    :' ID MSB
candata(2)=0    :'
candata(3)=0    :'
candata(4)=20   :' ID LSB
candata(5)="A"  :' CAN Data 1
candata(6)="B"  :' CAN Data 2
candata(7)="C"  :' CAN Data 3
candata(14)=3   :' CAN Length
candata(15)=0   :' CAN Mode
CANWRITE 1,candata(0)
```

Siehe auch:

`CANBAUD, CANERROR, CANID, CANREAD`

CASE *Anweisung*

Siehe `SELECT .. CASE`.

CHR$ *Funktion*

Liefert einen String, der aus dem Zeichen mit dem angegebenen ASCII-Code besteht.

Syntax:

```
s$=CHR$(n)
```
Beispiel:
```
PRINT CHR$(65)
```
Siehe auch:
```
ASC
```

CINT

Funktion

Der numerische Ausdruck `a` wird zu einer ganzen Zahl gerundet.

Syntax:
```
n=CINT(a)
```
Beispiel:
```
PRINT CINT(65.6)
```
Siehe auch:
```
ABS, FRAC, INT, FIX
```

CLEAR

Anweisung

Die Ereignisverfolgung wird ausgeschaltet (falls nicht vorher mit `CONTEVENTS ON` verhindert), nicht referenzierte Variablen werden gelöscht und der Speicher wird freigegeben, offene Dateien werden geschlossen. Referenzierte Variablen werden auf Standardwerte gesetzt.

Syntax:
```
CLEAR
```
Siehe auch:
```
CONTEVENTS, NEW
```

CLEARBUF

Anweisung

Der Eingangspuffer des betreffenden Kanals wird gelöscht.

Syntax:
```
CLEARBUF n
```
Siehe auch:
```
OPEN, INPUT
```

CLEARSTACK

Anweisung

Löscht den Basic-Stack.

Syntax:
```
CLEARSTACK
```
Siehe auch:
```
POPSTACK
```

CLOSE

Anweisung

Schließt eine oder mehrere offene Dateien.

Syntax:
```
CLOSE [[#]n1][,[#]n2]...]
```
Beispiel:
```
CLOSE
CLOSE #1,#2
```
Siehe auch:

OPEN

CLS

Anweisung (nur mit Display)
Löscht den Bildschirm, schaltet den Cursor ein und setzt ihn in die linke obere Ecke.
Syntax:
CLS
Siehe auch:
LOCATE, CURSOR

CODE(C)

Funktion
Liefert den ASCII-Code der zuletzt gedrückten Funktionstaste.
Syntax:
a=CODE(C)
Beispiel:
PRINT CODE(C)
Siehe auch:
CTRLCODE, ON xx GOTO/GOSUB

COM

Anweisung
Schaltet die Ereignisverfolgung der seriellen Schnittstelle n ein oder aus. Mit COM(n) STOP wird die Ereignisverfolgung angehalten, d.h. auftretende Ereignisse werden erst abgearbeitet, nachdem die Ereignisverfolgung wieder eingeschaltet wurde.
Syntax:
COM(n) ON|OFF|STOP
Siehe auch:
ON xx GOTO|GOSUB

CONNECT

Anweisung
Verbindet eine Slave-Steuerung mit der Mastersteuerung über die angebene Schnittstelle. Dabei wird der Schnittstelle die Nummer n zugewiesen, die beim Wechsel der Slavesteuerung verwendet wird (siehe NETMASTER Anweisung). n kann Werte zwischen 1 und 31 annehmen.
Syntax:
CONNECT n,s$
Beispiel:
CONNECT 1,"COM1:"
CONNECT 2,"CAN1:3"
Siehe auch:
NETMASTER, NETSLAVE

CONTEVENTS

Anweisung
Schaltet die Ereignisverfolgung nach dem Beenden eines Programmes ein oder aus. Bei eingeschalteter Ereignisverfolgung werden TIMER-, COM-, CAN- und andere Ereignisse weiterhin bearbeitet (d.h. es wird in die mit ON xx

GOSUB/GOTO angegebene Programmzeile gesprungen), auch wenn das Programm beendet ist. Das Programm darf nicht verändert werden, sonst kann es zu Abstürzen kommen. Die Verwendung von NEWLIST ist angeraten. Die Ereignisverfolgung wird auch bei den Befehlen CLEAR und NEW nicht abgeschaltet.

Syntax:
```
CONTEVENTS ON|OFF
```
Siehe auch:
```
CLEAR, NEW, ON xx GOTO|GOSUB
```

CONTRAST

Anweisung (nur mit Display)

Einstellung des LC-Display-Kontrastes. Werte von n zwischen –10 und 10 ändern den Kontrast relativ um den angegebenen Wert, Werte über 10 werden absolut übernommen. Der Maximalwert ist 100. Die Kontrasteinstellung kann mit der Anweisung WRITEPREFS dauerhaft bespeichert werden.

Syntax:
```
CONTRAST n
```
Beispiel:
```
CONTRAST -2
CONTRAST 70
```
Siehe auch:
```
WRITEPREFS
```

COS

Funktion

Liefert den Cosinus eines numerischen Ausdrucks.

Syntax:
```
a=COS(a)
```
Beispiel:
```
PRINT COS(x)
PRINT COS(0.4)
```
Siehe auch:
```
ACOS, ASIN, SIN
```

CTRLCODE

Anweisung (nur mit Tastatur)

Schaltet die Ereignisverfolgung für Tasten mit spezieller Funktion ein und aus. Der dem letzen Tastendruck entsprechende ASCII-Code kann mit der Funktion CODE(C) gelesen werden.

Taste	ASCII-Code
STOP	3
HALT	4
EDIT	5
START	6
RUN	7
MENU	14
MOVE	15
STEP	16
REF	18
TEACH	20
ESC	27

Syntax:
`CTRLCODE ON|OFF`
Siehe auch:
`ON xx GOTO|GOSUB, CODE`

CURSOR *Anweisung (nur mit Display)*
Schaltet den Cursor ein oder aus.
Syntax:
`CURSOR ON|OFF`
Siehe auch:
`CLS, LOCATE`

DELETE *Anweisung*
Löscht eine Datei.
Syntax:
`DELETE s$`
Beispiel:
`DELETE "TEST1.BAS"`
Siehe auch:
`OPEN, FORMAT`

DIM *Anweisung*
Deklariert und dimensioniert ein Datenfeld. Die Indizes `n1`, `n2` und `n3` geben die Größe des Feldes an. Als Variablentypen sind `INTEGER`, `FIX`, `CHAR` und `STRING` erlaubt.
Syntax:
```
DIM array(n1[,n2[,n3]]) AS INTEGER
DIM array(n1[,n2[,n3]]) AS FIX
DIM array(n1[,n2[,n3]]) AS CHAR
DIM array(n1[,n2[,n3]]) AS STRING
```
Beispiel:
```
DIM xcoord(100) AS FIX
DIM men1(5) AS STRING
DIM data(64,64,64) AS FIX
```

DO..LOOP *Anweisung*

Wiederholt einen Anweisungsblock solange wie eine Bedingung wahr ist (`WHILE a`) oder bis eine Bedingung wahr ist (`UNTIL a`). Je nach Position von `WHILE` und `UNTIL` findet die Überprüfung der Bedingung vor oder nach jedem Anweisungsblockdurchlauf statt. Der Ausdruck `a` wird als wahr (`TRUE`) interpretiert, wenn er ungleich Null ist, als falsch (`FALSE`), wenn er gleich Null ist.

Syntax:

```
DO [WHILE | UNTIL a]
  [Anweisungsblock]
LOOP [WHILE | UNTIL a]
```

Beispiel:

```
x=1
DO WHILE x<10
  x=x+1: PRINT x
LOOP
```

Siehe auch:

`EXIT, FOR..NEXT`

DRESTORE

Anweisung

Stellt den in Array `a(0)` zwischengespeicherten Bildschirminhalt wieder her.

Syntax:

```
DRESTORE a(0)
```

Siehe auch:

`CLS, DSAVE`

DSAVE

Anweisung

Speichert den aktuellen Bildschirminhalt im Array `a(0)` (`of char`) ab. Das Array muß so groß sein, daß alle Bildschirmzeichen Platz haben.

Syntax:

```
DSAVE a(0)
```

Siehe auch:

`CLS, DRESTORE`

ECHO

Anweisung

Schaltet den Echo-Modus (eingehende Zeichen werden wieder ausgegeben) einer Kommunikationsschnittstelle ein oder aus.

Syntax:

```
ECHO ON|OFF
ECHO s$,ON|OFF
```

Beispiel:

```
ECHO "AUX1:",ON
```

Siehe auch:

`PROMPT, SHELL, ASSIGN`

EDIT

Anweisung (nur mit Display)

Ruft den Texteditor auf. Falls eine Zeilennummer `n` angegeben ist, wird der Cursor an den Anfang dieser Zeile gesetzt. Wenn der Dateiname `s$` fehlt, startet der Editor mit dem **FILE/LOAD**-Menu. Ist statt eines Dateinamens nur ein Formatstring angegeben, werden nur die dem Format entsprechenden Dateien aufgelistet.

Syntax:

```
EDIT [s$[,n]]
```

Beispiel:

```
EDIT "TEST1.BAS"
EDIT name$,errorline
EDIT "*.BAS"
```

Siehe auch:

END *Anweisung*

Beendet ein Programm (`END`) oder einen Block (`END IF`, `END SELECT`). Beim Beenden eines Programmes werden alle offenen Dateien geschlossen.

Syntax:

```
END
END IF
END SELECT
```

Siehe auch:

```
IF..THEN..ELSE, SELECT..CASE
```

EOF *Funktion*

Liefert den Wert –1 (wahr), wenn das Ende der Datei `n` erreicht ist und den Wert 0 (falsch), wenn nicht.

Syntax:

```
a=EOF(n)
```

Beispiel:

```
OPEN "TEST.TXT" FOR INPUT AS #1
DO
  PRINT INPUT$(1,1);
LOOP UNTIL EOF(1)
```

Siehe auch:

```
OPEN
```

EOLCHAR *Anweisung*

Bestimmt das Zeichen, das als Zeilenende bei der `INPUT`, der `LINE INPUT`, der `TINPUT` und der `TINPUTL` Anweisung interpretiert wird. Voreingestellt ist Carriage-Return (ASCII-Code: 13).

Syntax:

```
EOLCHAR n
EOLCHAR s$
```

Beispiel:

```
EOLCHAR 10
EOLCHAR "*"
```

Siehe auch:

INPUT, LINE INPUT

ERL

Funktion
Liefert die Programmzeilennummer der Zeile, in der zuletzt ein Fehler aufgetreten ist.
Syntax:
a=ERL
Siehe auch:
ERR, ERR$

ERR

Funktion
Liefert den Fehlercode des letzten Fehlers.
Syntax:
a=ERR
Siehe auch:
ERL, ERR$

ERR$

Funktion
Liefert den Beschreibungstext zum Fehlercode n.
Syntax:
s$=ERR$(n)
Beispiel:
PRINT ERR$(ERR)
Siehe auch:
ERL, ERR

ERROR

Anweisung
Schaltet die Fehlerbehandlung ein oder aus.
Syntax:
ERROR ON|OFF
Siehe auch:
ON xx GOTO|GOSUB

EXEC

Anweisung
Führt die Anweisungen in einer Datei aus. Falls in der Datei Zeilen mit Programmzeilennummern vorkommen, werden diese zum Programm hinzugefügt, aber nicht direkt ausgeführt. Die Programmausführung kann mit dem Befehl EXIT beendet werden.
Syntax:
EXEC s$
Beispiel:
EXEC "TEST.BAS"
Siehe auch:
LOAD, SAVE, EXIT

EXECERRORS

Anweisung

Schaltet den Abbruch im Fehlerfalle einer mit EXEC gestarteten Dateiausführung ein oder aus. Dies ist nützlich, falls von einem Programm aus eine Programmdatei gestartet wird, die eventuell Fehler haben kann.

Syntax:

```
EXECERRORS ON|OFF
```

Siehe auch:

```
EXEC, FERL, FERR, EXECSTR
```

EXECSTR *Anweisung*

Führt die Anweisungen in einem String aus. Falls im String Zeilen mit Programmzeilennummern vorkommen, werden diese zum Programm hinzugefügt, aber nicht direkt ausgeführt. Dieser Befehl ist eine Möglichkeit, innerhalb eines Programmes G-Codes mit variablen Parametern auszuführen. Der String muß als Stringvariable übergeben werden, eine Stringkonstante (z.B. EXECSTR "G00 X0") ist nicht möglich.

Syntax:

```
EXECSTR s$
```

Beispiel:

```
a$="G01 X"+2*xofs
EXECSTR a$
```

Siehe auch:

```
EXEC
```

EXIT *Anweisung*

Beendet eine DO..LOOP oder eine FOR..NEXT-Schleife oder eine mit EXEC gestartete Dateiausführung vorzeitig. Hinweis: wird eine DO..LOOP oder eine FOR..NEXT-Schleife nicht mit EXIT, sondern mit GOTO verlassen, kann es zu einem Überlauf des BASIC-Stacks kommen.

Syntax:

```
EXIT DO
EXIT FOR
EXIT
```

Siehe auch:

```
DO..LOOP, FOR..NEXT, EXEC
```

FERL *Funktion*

Liefert die Textzeile der Datei, in der bei der Ausführung mit EXEC ein Fehler auftrat.

Syntax:

```
a=FERL
```

Beispiel:

```
PRINT FERL
```

Siehe auch:

```
EXEC, EXECERRORS, FERR
```

FERR *Funktion*

Liefert den Fehlercode des letzen bei der Ausführung mit `EXEC` aufgetretenen Fehlers.

Syntax:

```
a=FERR
```

Beispiel:

```
PRINT FERR
```

Siehe auch:

`EXEC, EXECERRORS, FERL`

FILES *Anweisung*

Zeigt das Inhaltsverzeichnis eines Datenträgers an. Falls kein Datenträgername angegeben wird, wird `SID1:` verwendet. Wird die Option `,1` hinter dem Dateinamen angegeben, werden auch versteckte Dateien (Endung `.SYS`) angezeigt. Die Verwendung von Wildcarts (*,?) im Dateinamen ist zulässig.

Syntax:

```
FILES [s$][,1]
```

Beispiel:

```
FILES
FILES "*.BAS"
FILES "*.*",1
```

Siehe auch:

`FORMAT, LOAD, SAVE, EXEC`

FIX *Funktion*

Wandelt einen numerischen Ausdruck `a` in eine Integerzahl um, indem der Hinterkommaanteil auf Null gesetzt wird.

Syntax:

```
n=FIX(a)
```

Beispiel:

```
PRINT FIX(65.6)
```

Siehe auch:

`ABS, CINT, FRAC, INT`

FOR..NEXT *Anweisung*

Wiederholt einen Anweisungsblock bis die Zählvariable `a` beginnend vom Startwert `astart` den Endwert `aende` erreicht oder überschritten hat. Die Schrittweite `aschritt` kann nach der `STEP`-Anweisung angegeben werden.

Syntax:

```
FOR a = astart TO aende [STEP aschritt]
  [Anweisungsblock]
NEXT [a]
```

Beispiel:

```
FOR i = 1 TO 100: PRINT i: NEXT i
FOR x = 100.5 TO 13.2 STEP -0.4
  PRINT x
NEXT
```

Siehe auch:

DO..LOOP, EXIT

FORMAT *Anweisung*

Formatiert einen Datenträger. Es muß ein Datenträgername angegeben werden, z.B. "SID1:". Alle Daten werden gelöscht. FLASH-Laufwerke müssen mit der UPDATE Anweisung aktualisiert werden.

Syntax:
FORMAT s$

Beispiel:
FORMAT "SID1:"

Siehe auch:
FILES, LOAD, SAVE, EXEC, UPDATE

FRAC *Funktion*

Gibt den 32 Bit Hinterkommaanteil einer Fix-Zahl als 32 Bit Integerzahl zurück.

Syntax:
n=FRAC(a)

Beispiel:
PRINT FRAC(65.75)

Siehe auch:
ABS, CINT, FIX, INT

FRE *Funktion*

Gibt Informationen über den verfügbaren Speicher zurück. FRE(0) gibt den freien Speicher, FRE(1) den größten zusammenhängenden Speicherblock und FRE(2) den verfügbaren Basicstack zurück.

Syntax:
n1=FRE(n2)

Siehe auch:
MEM, STACKSIZE, WRITEPREFS

GETFILENAME$ *Funktion (nur mit Display)*

Liest einen Dateinamen ein. Dabei muß der String title$ als Titel vorgegeben werden (z.B. "Datei Laden"), ein Suchstring sel$ zur Auswahl der anzuzeigenden Dateien, eine Flagvariable n und optional ein Default-String default$. Die Flagvariable n gibt an, ob zusätzlich zu den Dateinamen noch die Zeile "NEW FILE" angezeigt werden soll (n=0) oder nicht (n=1). Dies ermöglicht die Eingabe eines neuen Dateinamens, wobei default$ als vorgegebener Text erscheint. Dateien mit der Erweiterung SYS werden nicht angezeigt.

Syntax:
name$=GETFILENAME$(title$,sel$,n[,DEFAULT$])

Beispiel:
name$=GETFILENAME$("Load File","*.BAS",0)
n$=GETFILENAME$("SAVE","*.TXT",1,"NONAME.TXT")

Siehe auch:
MENU

GOSUB .. RETURN

Anweisung
Springt zu einem Unterprogramm in der Programmzeile n. Mit RETURN kann wieder unmittelbar hinter die GOSUB-Anweisung zurückgesprungen werden. Ein Unterprogramm sollte immer mit RETURN beendet werden, da es sonst zu einem Stackfehler kommen kann.

Syntax:
```
GOSUB n
```
Beispiel:
```
100  GOSUB 1000
110  END
1000 PRINT "Hallo!"
1010 RETURN
```
Siehe auch:
GOTO

GOTO

Anweisung
Das Programm wird an der Programmzeile n fortgesetzt.

Syntax:
```
GOTO n
```
Siehe auch:
GOSUB, RETURN

HEX$

Funktion
Liefert die n2-stellige Hexadezimaldarstellung einer Integerzahl n1 als String.

Syntax:
```
s$=HEX$(n1,n2)
```
Beispiel:
```
100 PRINT HEX$(10000,4)
```
Siehe auch:
BIN$, OCT$

IF..THEN..ELSE..END IF

Anweisung
Wertet den boolschen Ausdruck nb aus und führt die Anweisung nach THEN aus, wenn der Ausdruck wahr ist. Falls der Ausdruck falsch ist werden die Anweisungen nach ELSE ausgeführt. Der ELSE-Zweig kann auch entfallen.

Syntax:
```
IF nb THEN
  Anweisungsblock 1
ELSE
  Anweisungsblock 2
END IF
```

Beispiel:
```
IF a+b<>125 THEN PRINT ”1”
  ELSE PRINT ”2”
END IF
```
Siehe auch:
GOTO

INDENT *Anweisung*

Schaltet die automatische Formatierung des Programmlistings ein oder aus.

Syntax:
```
INDENT ON|OFF
```
Siehe auch:
LIST

INPUT *Anweisung*

Liest eine Variablenliste ein. Optional kann eine Kanalnummer `#n` angegeben werden. Wenn es sich um eine Kommunikationsschnittstelle handelt, wird bei einer Fehleingabe eine Neueingabe verlangt, sonst wird mit einer Fehlermeldung abgebrochen.

Syntax:
```
INPUT [;][”Text”{;|,}] var1[,var2[,var3..]]
INPUT #n,var1[,var2[,var3..]]
```
Siehe auch:
LINE INPUT, INPUT$, TINPUT, TINPUTC, TINPUTL

INPUT$ *Funktion*

Liest eine bestimmte Zahl `n1` von Zeichen aus einer Datei bzw. von einer Schnittstelle. Wenn keine Kanalnummer `#n2` angegeben wird, wird von der Standard-Kommunikationsschnittstelle (Tastatur und COM1) gelesen.

Syntax:
```
s$=INPUT$(n1[,[#]n2])
```
Siehe auch:
INPUT, INPUTF, LINE INPUT, TINPUT, TINPUTC, TINPUTL

INPUTF *Anweisung*

Liest eine Variablenliste ein. Optional kann eine Kanalnummer `#n` angegeben werden. In der Integervariable `flag` wird ein Fehlercode zurückgegeben, falls ein Fehler aufgetreten ist, und 0 falls nicht.

Syntax:
```
INPUTF flag,[”Text”{;|,}] var1[,var2..]
INPUTF #n,flag,var1[,var2[,var3..]]
```
Siehe auch:
LINE INPUT, INPUT, INPUT$

INSTR

Funktion

Gibt die Position des ersten Auftretens der Zeichenkette `s2$` in der Zeichenkette `s1$` an. Dabei wird mit der Suche ab der optional anzugebenden Position `n2` begonnen. Statt `s2$` kann auch ein ASCII-Code angegeben werden.

Syntax:

```
n1=INSTR([n2,]s1$,s2$)
n1=INSTR([n2,]s1$,n)          ' n=1..255
```

Siehe auch:

`INPUT, LINE INPUT`

INT

Funktion

Liefert die größte Integerzahl, die kleiner oder gleich dem numerischen Ausdruck `a` ist.

Syntax:

```
n=INT(a)
```

Siehe auch:

`ABS, CINT, FIX, FRAC`

KEY

Anweisung

Schaltet die Ereignisverfolgung für das Tastenfeld ein oder aus.

Syntax:

```
KEY ON|OFF
```

Siehe auch:

`ON xx GOTO|GOSUB`

KEYTIME

Funktion

Gibt an, wie lange die gerade gedrückte Taste schon gedrückt wurde. Der Integerwert gibt die Zeit in 1/1024 s an.

Syntax:

```
n=KEYTIME
```

Siehe auch:

`KEY, KEYREPEAT`

KEYREPEAT

Anweisung

Setzt die Parameter der Tastenwiederholfunktion. `a1` gibt die Verzögerung bis zu Wiederholung in Sekunden an, `a2` die Verzögerung zwischen den wiederholten Zeichen, `a3` die Verzögerung zwischen den wiederholten Zeichen im Schnellmodus und `n` die Zahl der Zeichen, ab der der Schnellmodus eingeschaltet wird. `a4` gibt an, wie lange das Relais der **START**-Taste mindestens geschlossen bzw. das Relais der **STOP**-Taste mindestens geöffnet wird, wenn die Tasten betätigt werden (mit der Tastatur oder den Befehlen `START` und `NOTAUS`).

Die Parameter können mit der Anweisung `WRITEPREFS` dauerhaft bespeichert werden.

Syntax:
`KEYREPEAT a1,a2,a3,n,a4`
Siehe auch:
`KEY, KEYTIME, START, NOTAUS`

LCASE$ *Funktion*

Wandelt die Buchstaben einer Zeichenfolge in Kleinbuchstaben um.
Syntax:
`s$=LCASE$(s$)`
Siehe auch:
`UCASE$`

LED *Anweisung*

Schaltet die Leuchtdioden des Tastenfeldes ein, aus oder in den Blink-Modus. Dabei ist `n=1` die LED der Taste **REF**, `n=2` Taste **TEACH**, `n=3` Taste **EDIT**, `n=4` Taste **STEP**. Die anderen LEDs werden automatisch geschaltet.
Syntax:
`LED [n1][,n2[,n3..]] ON|OFF|BLINK`

LEFT$ *Funktion*

Gibt die ersten `n` Zeichen der Zeichenkette `s$` zurück.
Syntax:
`s$=LEFT$(s$,n)`
Siehe auch:
`LEN, MID$, RIGHT$`

LEN *Funktion*

Liefert die Länge der Zeichenkette `s$`.
Syntax:
`n=LEN(s$)`
Siehe auch:
`LEFT$, MID$, RIGHT$`

LET *Anweisung*

Weist einer Variablen einen Wert zu. Da der Befehl nicht notwendig ist, wir er im Listing nicht angezeigt.
Syntax:
`LET varname=value : ' gleichbedeutend mit varname=value`

LINE INPUT *Anweisung*

Liest eine Zeile ein. Optional kann eine Kanalnummer `#n` angegeben werden.
Syntax:
`LINE INPUT [;]["Text";] s$`
`LINE INPUT #n,s$`

Siehe auch:
`INPUT, INPUT$, INPUTF, TINPUT, TINPUTC, TINPUTL`

LIST *Anweisung*

Gibt das Programmlisting aus. Optional kann eine Startzeilennummer und ein Endzeilennummer angegeben werden.

Syntax:
`LIST [n1][-n2]`

Siehe auch:
`IDENT`

LOAD *Anweisung*

In der vorliegenden Version gleichbedeutend mit `EXEC`.

Syntax:
`LOAD s$`

Siehe auch:
`EXEC`

LOC *Funktion*

Liefert die aktuelle Schreib/Lese-Position innerhalb der Datei `n`.

Syntax:
`n=LOC(n)`

Siehe auch:
`CLOSE, LOF, OPEN, SEEK`

LOCATE *Anweisung (nur mit Display)*

Setzt den Cursor an die angegebene Position `n1,n2`.

Syntax:
`LOCATE n1,n2`

Siehe auch:
`CURSOR`

LOF *Funktion*

Liefert die aktuelle Größe der Datei `n`.

Syntax:
`n=LOF(n)`

Siehe auch:
`CLOSE, LOC, OPEN, SEEK`

LTRIM$ *Funktion*

Löscht alle Leerzeichen, Tabulatoren und Kontrollzeichen am Anfang der Zeichenkette `s$`.

Syntax:
`s$=LTRIM$(s$)`

Siehe auch:
`RTRIM$`

MEM *Anweisung*

Gibt Informationen zur Speichernutzung aus.

Syntax:

```
MEM
```

Siehe auch:

```
FRE
```

MENU *Funktion (nur mit Display)*

Gibt die Nummer `n` eines ausgewählten Menüpunktes eines vordefinierten Zeichenkettenarrays `men(0)` zurück. Falls das Menü mit **ESC** abgebrochen wird, wird der Wert 0 zurückgegeben. Die Zeichenkette `titel$` erscheint als Überschrift in der ersten Zeile.

Syntax:

```
n=MENU(titel$,men(0))
```

Beispiel:

```
DIM men1(3) AS STRING
men1(1)="LOAD"
men1(2)="SAVE"
men1(3)="EXIT"
sel=MENU("FILE",men1(0))
```

Siehe auch:

```
DIM, MESSAGE
```

MESSAGE *Anweisung (nur mit Display)*

Gibt einen Text oder eine Fehlermeldung aus, die mit einem Tastendruck bestätigt werden muß. Als Parameter wird entweder eine Zeichenkette `s$` oder die Nummer des Fehlercodes `n` angegeben.

Syntax 1:

```
MESSAGE n
```

Syntax 2:

```
MESSAGE s$
```

Siehe auch:

```
MENU
```

MID$ *Funktion*

Gibt `n2` Zeichen der Zeichenkette `s$` ab Position `n1` zurück. Fehlt der Parameter `n2`, wird die Zeichenfolge bis zum Ende von Zeichenkette `s$` zurückgegeben.

Syntax:

```
s$=MID$(s$,n1[,n2])
```

Siehe auch:

```
LEFT$, LEN, RIGHT$
```

MODE *Anweisung*

Setzt die Parameter der seriellen Schnittstellen und erlaubt deren Überprüfung. Die maximale Baudrate und die genauen Werte sind von der Ausstattung abhängig. Es können sich

durch Rundung gewisse Abweichungen ergeben, die die Kommunikation jedoch nicht beeinflussen. Als Parität sind die Schlüsselwörter NONE, EVEN, ODD, MARK und SPACE erlaubt. Die Anzahl der Bits kann je nach Ausstattung 5 bis 8 betragen, die Anzahl der Stopbits 1 oder 2. Als Handshaketyp kann NONE, RTSCTS oder XONXOFF verwendet werden.

Syntax 1:

```
MODE COMn:baudrate,parity,bits,stop,handshake
```

Syntax 2:

```
MODE COMn:
```

Beispiel:

```
MODE COM1:19200,NONE,8,1,XONXOFF
```

NETMASTER *Anweisung*

Der Parameter ON schaltet den Steuerung in den Netzwerk-Master Modus. Dies bedeutet, daß bei der Eingabe von Befehlen in der Kommandozeile die Möglichkeit besteht, zu der Kommandozeile einer anderen Steuerung innerhalb des Netzwerkes zu wechseln. Die anderen Steuerungen können über serielle Schnittstellen oder den CAN-Bus angeschlossen sein (siehe CONNECT Anweisung). Die aktuelle Zielsteuerung ist am Buchstaben bzw. an der Zahl vor dem Prompt-Zeichen > erkennbar. Ein M wird angezeigt, solange die Befehle an die Master-Steuerung gerichtet sind. Die Kommunikation mit einer Slave-Steuerung ist am vorangestellten S bzw. der Nummer der Slavesteuerung (siehe NETSLAVE Anweisung) erkennbar, sofern die Steuerung schon im Slave-Modus ist.
Zwischen den Steuerungen kann gewechselt werden, indem eine Kommandozeile mit dem Zeichen @ und der Nummer der gewünschten Steuerung begonnen wird. Weitere Anweisungen in der Zeile werden ignoriert. Die Mastersteuerung hat die Nummer 0, alle anderen die mit der CONNECT Anweisung zugewiesene Nummer.
Der Master-Modus wird mit dem Parameter OFF wieder ausgeschaltet.

Beispiel:

```
>NETMASTER ON
M>CONNECT 2,"COM1:"
M>@2
S>
```

Syntax:

```
NETMASTER ON|OFF
```

Siehe auch:

```
CONNECT, NETSLAVE
```

NETSLAVE *Anweisung*

Schaltet den Netzwerk-Slave Modus ein oder aus. `s$` gibt die Kommunikationsschnittstelle an (seriell oder CAN), der optionale Parameter `n` die Slavenummer. Wird er weggelassen, wird bei der Mastersteuerung nur ein `S` vor dem Prompt angezeigt, sonst die angegebene Slavenummer. Die Slavenummer sollte identisch mit der bei der `CONNECT` Anweisung zugewiesenen Nummer sein.
Der Befehl hat eine ähnliche Wirkung wie die Zuweisung der angegebenen Schnittstelle zu `STD:` mit der `ASSIGN` Anweisung.

Syntax:

```
NETSLAVE OFF
NETSLAVE s$[,n]
```

Beispiel:

```
NETSLAVE "COM1:",12
NETSLAVE "CAN1:10",10
```

Siehe auch:

`CONNECT, NETMASTER`

NEW

Anweisung

Die Ereignisverfolgung wird ausgeschaltet (falls nicht vorher mit `CONTEVENTS ON` verhindert), ein im Speicher befindliches Programm wird gelöscht und der von Variablen belegten Speicher wird frei gegeben.

Syntax:

```
NEW
```

Siehe auch:

`CLEAR, CONTEVENTS, RENEW`

NEWLIST

Anweisung

Das vorhandene Programm wird für Zugriffe gesperrt, es kann ein neues Programm begonnen werden. Das alte Programm wird nur noch bei der Ereignisverfolgung verwendet. Es ist somit möglich, ein Anwendungsprogramm zu schreiben, während die Ereignisverfolgung unverändert im Hintergrund weiterläuft.

Syntax:

```
NEWLIST
```

Siehe auch:

`CLEAR, CONTEVENTS, NEW, ON xx GOTO|GOSUB, RENEW`

OCT$

Funktion

Liefert die `n2`-stellige Oktaldarstellung einer Integerzahl `n1` als String.

Syntax:

```
s$=OCT$(n1,n2)
```

Beispiel:

```
100 PRINT OCT$(10000,8)
```

Siehe auch:
`BIN$, HEX$`

ON

Anweisung
Stellt die Programmzeile `n` ein, zu der bei dem Ereignis `event` gesprungen werden soll. Es sind folgende Ereignisse möglich:

`CAN(n)`
Es wurde ein CAN-Frame im Kanal `n` empfangen.

`COM(n)`
Es wurde ein Zeichen von der seriellen Schnittstelle `n` eingelesen.

`CTRLCODE`
Eine Taste spezieller Funktion (**MENU**, **ESC**, **TEACH** ..) wurde gedrückt, bzw. der entsprechende Code wurde über die serielle Schnittstelle eingelesen.

`ERROR`
Es ist ein Laufzeitfehler (z.B. Division durch Null) aufgetreten.

`KEY`
Eine Taste der Frontplattentastatur wurde gedrückt.

`PORT(n)`
Am Eingang `n` (`n=1..8 (16)`) wurde die mit `PORT(n) xx` programmierte Flanke detektiert.

`TIMER(n)`
Dieses Ereignis wird alle `n` Millisekunden ausgelöst.

Syntax:
`ON event GOTO|GOSUB n`

Siehe auch:
`COM, CTRLCODE, ERROR, KEY, PORT, TIMER`

OPEN

Anweisung
Öffnet eine Datei. Dateiname und Gerät werden in `name$` angegeben. Wird der Gerätename weggelassen, wird `SID1:` verwendet. Bei Geräten ohne Dateisystem (z.B. `COM1:`) darf kein Dateiname verwendet werden. Als `modus` ist möglich:

`INPUT`
Aus der Datei kann nur gelesen werden.

`OUTPUT`
Die Datei kann nur geschrieben werden. Falls die Datei schon existiert, wird sie gelöscht.

`RANDOM`
Die Datei kann gelesen und geschrieben werden. Falls die Datei schon existiert, wird sie gelöscht.

`APPEND`

Die Datei kann gelesen und geschrieben werden. Eine existierende Datei wird nicht gelöscht, der Zeiger wird auf das Dateiende gesetzt.
Falls der optionale Parameter `FLAG` existiert, wird das Programm nicht abgebrochen, wenn ein Fehler beim Öffnen einer Datei aufgetreten ist, sondern es wird der Fehlercode in der Variable `n2` zurückgegeben.
Folgende Geräte werden abhängig von der Hardware-ausstattung unterstützt:

`COM1:`	erste serielle Schnittstelle
`COM2:`	zweite serielle Schnittstelle (je nach Baureihe)
`COM3:`	dritte serielle Schnittstelle (je nach Baureihe)
`COM4:`	vierte serielle Schnittstelle (je nach Baureihe)
`COM5:`	fünfte serielle Schnittstelle (je nach Baureihe)
`DIS:`	Display
`KEY:`	Tastatur
`SID1:`	EEPROM-Disk
`SID0:`	FLASH-Disk (siehe `UPDATE`)
`STD:`	Standard Ein- (`COM1:` und `KEY:`) und Ausgabe (`COM1:` und `DIS:`).
`AUX1:`	Hilfsgerät zur Verwendung mit `ASSIGN`
`AUX2:`	Hilfsgerät zur Verwendung mit `ASSIGN`
`AUX3:`	Hilfsgerät zur Verwendung mit `ASSIGN`
`AUX4:`	Hilfsgerät zur Verwendung mit `ASSIGN`
`CAN1:`	CAN-Bus

Der CAN-Bus kann als serielle Verbindung zwischen einem Master und einem oder mehreren Slaves verwendet werden. Dazu müssen die Nachrichten-Identifier mit der `CANID 0,..` Anweisung eingestellt werden. Innerhalb der seriellen CAN-Kommunikation sind 31 logische Kanäle eingerichtet, die normalerweise jeweils einem Slave zugeordnet werden. Die Kanalnummer wird wie ein Dateiname nach dem Doppelpunkt angegeben, z.B. `"CAN1:21"`.

Syntax:

```
OPEN name$ FOR modus AS [#]n1 [FLAG n2]
```

Beispiel:

```
OPEN "COM1:" FOR RANDOM AS #1
OPEN "SID0:CONFIG.SYS" FOR OUTPUT AS #2
OPEN "TEST.TXT" FOR INPUT AS #3
OPEN "CAN1:7" FOR RANDOM AS #4
```

Siehe auch:

`CANID, CLOSE, MODE, OPENSTR, UPDATE`

OPENSTR *Anweisung*

Öffnet die Zeichenkette `s$` als Datei.

Syntax:

```
OPENSTR s$ AS [#]n
```

Siehe auch:

`CLOSE, OPEN`

PARITY

Funktion

Berechnet die Parität einer ganzen Zahl. Die Rückgabe ist 0 für gerade Parität und –1 für ungerade Parität.

Syntax:

```
n=PARITY(n)
```

PI

Funktion

Gibt die Zahl π im Fixkommaformat zurück.

Syntax:

```
a=PI
```

PININ

Funktion

Liefert den Status des Eingangspin `n` (siehe Anhang). Je nach Ausstattung darf `n` zwischen 1 und 40 liegen. Eine logische 1 (`TRUE`, bzw. -1) entspricht einem 24 V-Pegel, eine logische 0 (`FALSE`, bzw. 0) einem 0 V-Pegel.

Syntax:

```
n=PININ(n)
```

Beispiel:

```
PRINT PININ(12)
```

Siehe auch:

```
PINOUT, PORTIN, PORTOUT, PORT
```

PINOUT

Anweisung

Setzt den Ausgang `n1` auf den Wert `n2`. Je nach Ausstattung darf `n1` zwischen 1 und 24 liegen. Eine logische 1 (`TRUE`, bzw. -1) entspricht einem 24 V-Pegel, eine logische 0 (`FALSE`, bzw. 0) einem 0 V-Pegel.

Syntax:

```
PINOUT n1,n2
```

Beispiel:

```
PINOUT 1,1
```

Siehe auch:

```
PININ, PORTIN, PORTOUT, PORT
```

POPSTACK

Anweisung

Löscht das oberste Element des Basic-Stack. Ein Stackelement wird z.B. bei einer `DO`-Anweisung, einer `FOR`-Anweisung oder einer `GOSUB`-Anweisung erzeugt. Wird dieses Element nicht von den entsprechenden Anweisungen entfernt (z.B. `LOOP`, `NEXT`, `RETURN`), muß dies mit der `POPSTACK`-Anweisung geschehen, da es sonst zu einem Stacküberlauf kommen kann.

Syntax:

```
POPSTACK
```

Siehe auch:

```
CLEARSTACK, FRE, STACKSIZE
```

PORT *Anweisung*

Schaltet die Ereignisverfolgung der Eingangsflankenüberwachung ein oder aus. `PORT(n) RISING` detektiert eine steigende Flanke am Eingang `n`, `PORT(n) FALLING` eine fallende Flanke. Die Flankenerkennung ist nur bei den Eingängen 1 bis 8 möglich, je nach Steuerungsmodell auch bis 16.

Syntax:

```
PORT(n) RISING|FALLING|OFF
```

Siehe auch:

```
ON xx GOTO|GOSUB
```

PORTIN *Funktion*

Liefert den Status des 8-Bit-Eingangsports `n`. Je nach Ausstattung darf `n` zwischen 1 und 5 liegen. Eine logische 1 entspricht einem 24 V-Pegel, eine logische 0 einem 0 V-Pegel.

Syntax:

```
n=PORTIN(n)
```

Beispiel:

```
PRINT BIN$(PORTIN(1),8)
```

Siehe auch:

```
PININ, PINOUT, PORTOUT, PORT
```

PORTOUT *Anweisung*

Setzt die Ausgänge des Port `n1` auf den Wert `n2`. Je nach Ausstattung darf `n1` zwischen 1 und 3 liegen. Eine logische 1 entspricht einem 24 V-Pegel, eine logische 0 einem 0 V-Pegel.

Syntax:

```
PORTOUT n1,n2
```

Beispiel:

```
PORTOUT 1,&b10011100
```

Siehe auch:

```
PININ, PINOUT, PORTIN, PORT
```

PRINT *Anweisung*

Gibt Zeichenketten oder Zeichenkettenrepräsentationen von numerischen Ausdrücken auf der Standard-Ausgabeschnittstelle (`COM1:` und Display) oder in eine angegebene Dateinummer `n` aus. Ausdrücke, die in der Ausdrucksliste mit einem Semikolon getrennt werden, werden unmittelbar nacheinander ausgegeben, mit einem Komma getrennte Ausdrücke werden durch ein Tabulatorzeichen (ASCII-Code 9) getrennt. Die Ausgabe wird mit einem CR/LF (ASCII-Codes 13 und 10) abgeschlossen, falls am Ende der `PRINT`-Anweisung kein Semikolon oder Komma steht.

Syntax:

```
PRINT [#n,][a1|s1$][;|,]..
```

Beispiel:

```
PRINT ”HALLO”;CHR$(33)
```

```
PRINT 100.0,20;
PRINT #1,"TEST"
```

Siehe auch:
`PRINT USING, WRITE, XPRINT`

PRINT USING *Anweisung*

Gibt Zeichenketten oder Zeichenkettenrepräsentationen von numerischen Ausdrücken formatiert auf der Standard-Ausgabeschnittstelle (COM1: und Display) oder in eine angegebene Dateinummer `n` aus. Ausdrücke, die in der Ausdrucksliste mit einem Semikolon getrennt werden, werden unmittelbar nacheinander ausgegeben, mit einem Komma getrennte Ausdrücke werden durch ein Tabulatorzeichen (ASCII-Code 9) getrennt. Die Ausgabe wird mit einem CR/LF (ASCII-Codes 13 und 10) abgeschlossen, falls am Ende der `PRINT`-Anweisung kein Semikolon oder Komma steht. Für die Format-Zeichenkette `format$` gelten folgende Konventionen:

`###.##`	Anzahl der Vor- und Hinterkommastellen. Führende Leerstellen werden mit Spaces gefüllt.
`~.#`	Anzahl der Vorkommastellen beliebig
`$##.##`	Führende Leerstellen werden mit Nullen gefüllt
`+##.##`	Vorzeichen wird angezeigt
`&`	Druckt gesamte Zeichenkette
`!`	Druckt erstes Zeichen der Zeichenkette
`_`	Druckt nachfolgendes Sonderzeichen (z.B. `#`)
`\ \`	Druckt n Zeichen einer Zeichenkette n = (Anzahl der Leerzeichen zwischen `\\`) + 2

Syntax:
`PRINT [#n,] USING format$; [a1|s1$][;|,]..`
Beispiel:
`PRINT USING "X: ###.##";SIN(PI/3)`
Siehe auch:
`PRINT, WRITE`

PROMPT *Anweisung*

Schaltet die Ausgabe der Eingabeaufforderung `>` ein oder aus.
Syntax:
`PROMPT ON|OFF`
Siehe auch:
`ECHO, SHELL`

PULSEOUT *Anweisung*

Schaltet den Pulse Out Ausgang `n` ein oder aus. Die Angabe von `period` und `hightime` schaltet den Ausgang ein. `period` und `hightime` werden in Millisekunden angegeben und können Integer oder Fix Werte sein. Die Angabe von 0 für `period` schaltet den Ausgang aus.
Syntax:

```
PULSEOUT n,period,hightime      ' switch on
PULSEOUT n,0                    ' switch off
```

Beispiel:

```
PULSEOUT 1,20.0,1.5
```

RANDOMIZE *Anweisung*

Initialisiert den Pseudo-Zufallszahlen-Generator mit der ganzen Zahl `n`.

Syntax:

```
RANDOMIZE n
```

Siehe auch:

```
RND
```

REM *Anweisung*

Behandelt nachfolgende Zeichen der Programmzeile als Kommentar. `REM` kann auch mit dem Hochkomma ‘ abgekürzt werden.

Syntax:

```
REM
```

RENEW *Anweisung*

Löscht ein Benutzerprogramm, das während der Laufzeit des Hauptprogrammes eingegeben wurde. Das Hauptprogramm muß vorher mit `NEWLIST` geschützt werden.

Syntax:

```
RENEW
```

Siehe auch:

```
NEW, NEWLIST
```

RENUMBER *Anweisung*

Numeriert die Programmzeilen eines im Speicher befindlichen Programms neu. Dabei wird mit der Zeilennummer `n1` begonnen, `n2` ist die Schrittweite.

Syntax:

```
RENUMBER n1,n2
```

RESET *Anweisung*

Löst einen Hardware-Reset aus.

Syntax:

```
RESET
```

RIGHT$ *Funktion*

Gibt die letzten `n` Zeichen der Zeichenkette `s$` zurück.

Syntax:

```
s$=RIGHT$(s$,n)
```

Siehe auch:

```
LEN, LEFT$, MID$
```

RND *Funktion*

Gibt eine 32-Bit Pseudo-Zufallszahl zurück.

Syntax:

```
n=RND
```

Siehe auch:

```
RANDOMIZE
```

ROTATE *Funktion*

Rotiert eine 32-Bit-Zahl `n1` um `n2` Stellen.

Syntax:

```
n=ROTATE(n1,n2)
```

Siehe auch:

```
ASHIFT, SHIFT
```

RTRIM$ *Funktion*

Löscht alle Leerzeichen, Tabulatoren und Kontrollzeichen am Ende der Zeichenkette `s$`.

Syntax:

```
s$=RTRIM$(s$)
```

Siehe auch:

```
LTRIM$
```

RUN *Anweisung*

Startet die Ausführung eines Programmes, optional ab der Zeilennummer `n`.

Syntax:

```
RUN [n]
```

Siehe auch:

```
END
```

SAVE *Anweisung*

Speichert ein Programm unter dem angegebenen Namen. Falls kein Gerätenamen angegeben wird, wird `SID1:` verwendet. FLASH-Laufwerke müssen mit der `UPDATE` Anweisung aktualisiert werden.

Syntax:

```
SAVE name$
```

Siehe auch:

```
EXEC, FORMAT, LOAD, OPEN, UPDATE
```

SEEK *Anweisung*

Setzt die Schreib/Leseposition einer Datei `n1` auf Position `n2`.

Syntax:

```
SEEK [#]n1,n2
```

Siehe auch:

```
OPEN
```

SELECT .. CASE

Anweisung

Führt in Abhängigkeit von einem Testausdruck einen von mehreren Anweisungsblöcken aus. Als Vergleichsoperator `cmp` kann =, >, <, >=, <= oder <> verwendet werden.

Syntax:

```
SELECT CASE testausdruck
  CASE ausdruck: Anweisungsblock
  CASE IS cmp ausdruck: Anweisungsblock
  CASE ausdruck TO ausdruck: Anw.-block
  CASE ELSE Anweisungsblock
END SELECT
```

Beispiel:

```
FOR i=1 TO 5
  SELECT CASE i
    CASE 1: PRINT "1"
    CASE 2 TO 5: PRINT "2..5"
    CASE IS >= 6: PRINT ">=6"
  END SELECT
NEXT i
```

Siehe auch:

`IF..THEN`

SGN

Funktion

Liefert das Vorzeichen eines numerischen Ausdrucks.

Syntax:

```
n=SGN(a)
```

Beispiel:

```
PRINT SGN(-3)
PRINT SGN(0.4)
```

Siehe auch:

`ABS`

SHELL

Anweisung

Schaltet den Befehlszeileneditor, den Echo-Modus und die EIngabeaufforderung ein oder aus.

Syntax:

```
SHELL ON|OFF
```

Siehe auch:

`ECHO, PROMPT`

SHIFT

Funktion

Verschiebt eine 32-Bit-Zahl `n1` logisch bitweise um `n2` Stellen.

Syntax:

```
n=SHIFT(n1,n2)
```

Siehe auch:

`ASHIFT, ROTATE`

SIN

Funktion

Liefert den Sinus eines numerischen Ausdrucks.

Syntax:

```
a=SIN(a)
```
Beispiel:
```
PRINT SIN(x)
PRINT SIN(0.4)
```
Siehe auch:
```
ACOS, ASIN, COS
```

SLEEP

Anweisung
Wartet `n` Millisekunden. Die Ereignisverfolgung bleibt unbeeinflußt.
Syntax:
```
SLEEP n
```
Siehe auch:
```
TIME
```

SPACE$

Funktion
Erzeugt eine Zeichenkette aus `n` Leerzeichen.
Syntax:
```
s$=SPACE$(n)
```
Siehe auch:
```
STRING$
```

SQR

Funktion
Liefert die Quadratwurzel eines numerischen Ausdrucks.
Syntax:
```
a=SQR(a)
```

STACKSIZE

Anweisung
Stellt die Größe des BASIC-Stacks auf `n` Bytes. Die Änderung wird erst nach einem Neustart wirksam. Die Einstellung muß zuvor mit dem Befehl `WRITEPREFS` dauerhaft gespeichert werden.
Syntax:
```
STACKSIZE n
```
Siehe auch:
```
WRITEPREFS
```

STR$

Funktion
Gibt die Zeichenfolgendarstellung eines numerisches Ausdrucks als Zeichenkette zurück.
Syntax:
```
s$=STR$(a)
```
Siehe auch:
```
VAL
```

STRING$

Funktion

Gibt eine Zeichenkette zurück, die aus `n1` Zeichen mit dem ASCII-Code `n2` bzw. dem ersten Zeichen des Strings `s$` zusammengesetzt ist.

Syntax:

```
s$=STRING$(n1,n2|s$)
```

Siehe auch:

`SPACE$`

SYNC

Anweisung

Synchronisiert die Programmausführung mit dem Systemzeitgeber. Die Programmausführung wird unterbrochen, bis der Wert des durchlaufenden Systemzeitgebers durch n teilbar ist. Während der Wartezeit werden auch Ereignisse (Timer etc.) nicht verfolgt. Dieser Befehl sollte daher nur in besonderen Fällen eingesetzt werden. Beim Zählerüberlauf nach 1 000 000 000 Millisekunden kann es kurzzeitig falschen Wartezeiten kommen. Die Standardfrequenz des Systemzeitgebers beträgt 1024 Hz.

Syntax:

```
SYNC n
```

Siehe auch:

`SLEEP, SYSTIMER`

SYSTIMER

Anweisung (nur bei -m Versionen)

Stellt das Intervall des Systemzeitgebers auf `a` Millisekunden ein. Dieser Befehl ist nur in speziellen Betriebssystemversionen implementiert. Mit diesem Befehl kann die Frequenz des Systemzeitgebers z.B. auf 1000 Hz statt der standardmäßigen 1024 Hz eingestellt werden. Die Zeitauflösung beträgt etwa 200 ns. Die Veränderung des Systemzeitgebers bedeutet, daß zeitabhängige Berechungen wie z.B. Beschleunigungen oder Geschwindigkeiten abweichende Resultate liefern, `SLEEP` und `TIME` liefern jedoch korrekte Ergebnisse. Der Einsatz ist deshalb nur für Sonderfälle empfohlen. Die Intervallzeit sollte je nach Steuerungshardware und deren Auslastung 0.8 ms nicht unterschreiten und insbesondere bei Servomotoren 2 ms nicht überschreiten.

Syntax:

```
SYSTIMER a
```

Siehe auch:

`SLEEP, SYNC`

TIME

Funktion

Gibt die aktuelle Zeit seit dem Einschalten der Steuerung in Millisekunden zurück. Nach 1 000 000 000 Millisekunden springt der Zähler wieder auf 0.

Syntax:

```
n=TIME
```

Siehe auch:
SLEEP

TIMER *Anweisung*

Schaltet die Timer-Ereignisverfolgung ein oder aus.
Syntax:
```
TIMER ON|OFF|STOP
```
Siehe auch:
```
ON xx GOTO|GOSUB
```

TINPUT *Anweisung*

Liest eine Variablenliste über den Kanal `n` ein. Ist `n=0`, wird der Standard-Eingabekanal verwendet. Der Wert `timeout` gibt die maximale Wartezeit in Millisekunden an, innerhalb der die vollständige Eingabe abgeschlossen sein muß.
Ein Wert von `0` schaltet die Zeitüberwachung aus.
Die Variable `flag` enthält nach der Abarbeitung der Anweisung den Wert `0`, wenn die Eingabe erfolgreich war, und den Wert `-1`, wenn ein Fehler aufgetreten ist.
Syntax:
```
TINPUT n,flag,timeout,var1[,var2[,var3..]]
```
Beispiel:
```
TINPUT 2,iflag,2000,a,b
TINPUT kanal,iflag,0,a$
```
Siehe auch:
```
LINE INPUT, INPUT$, INPUT, TINPUTL, TINPUTC
```

TINPUTC *Anweisung*

Liest eine Zeichenkette der Länge `count` in die Stringvariable `string$` über den Kanal `n` ein. Ist `n=0`, wird der Standard-Eingabekanal verwendet. Der Wert `timeout` gibt die maximale Wartezeit in Millisekunden an, innerhalb der die vollständige Eingabe abgeschlossen sein muß.
Ein Wert von `0` schaltet die Zeitüberwachung aus.
Die Variable `flag` enthält nach der Abarbeitung der Anweisung den Wert `0`, wenn die Eingabe erfolgreich war, und den Wert `-1`, wenn ein Fehler aufgetreten ist.
Syntax:
```
TINPUTC n,flag,timeout,count,string$
```
Beispiel:
```
TINPUTC 2,iflag,2000,10,b$
TINPUTC kanal,iflag,0,laenge,a$
```
Siehe auch:
```
LINE INPUT, INPUT$, INPUT, TINPUTL, TINPUT
```

TINPUTL *Anweisung*

Liest eine Zeile (beendet mit CR bzw. mit dem über EOLCHAR vorgegebenen Zeichen) über den Kanal n ein. Ist n=0, wird der Standard-Eingabekanal verwendet. Der Wert timeout gibt die maximale Wartezeit in Millisekunden an, innerhalb der die vollständige Eingabe abgeschlossen sein muß.
Ein Wert von 0 schaltet die Zeitüberwachung aus.
Die Variable flag enthält nach der Abarbeitung der Anweisung den Wert 0, wenn die Eingabe erfolgreich war, und den Wert -1, wenn ein Fehler aufgetreten ist.

Syntax:
```
TINPUTL n,flag,timeout,a$
```
Beispiel:
```
TINPUTL 2,iflag,2000,a$
TINPUTL kanal,iflag,0,a$
```
Siehe auch:
LINE INPUT, INPUT$, INPUT, TINPUTL, TINPUTL

TYPE

Anweisung

Gibt den Inhalt der Datei mit Namen name$ auf dem Standard-Ausgabegerät aus.

Syntax:
```
TYPE name$
```
Siehe auch:
CLOSE, OPEN

UCASE$

Funktion

Wandelt die Buchstaben einer Zeichenfolge in Großbuchstaben um.

Syntax:
```
s$=UCASE$(s$)
```
Siehe auch:
LCASE$

UPDATE

Anweisung

Aktualisiert ein FLASH-Laufwerk nach einer SAVE oder FORMAT-Anweisung oder einer anderen Dateioperation.

Syntax:
```
UPDATE s$
```
Beispiel:
```
UPDATE "SID0:"
```
Siehe auch:
FILES, LOAD, SAVE, EXEC, FORMAT

VAL

Funktion

Wandelt eine Zeichenfolge in ein Zahl um.

Syntax:
```
a=VAL(s$)
```
Siehe auch:
STR$

VERSION$ *Funktion*

Gibt die Softwareversion als String zurück.

Syntax:

```
s$=VERSION$
```

WRITE *Anweisung*

`WRITE` gibt im Gegensatz zu `PRINT` die Ausdrucksliste in einer Form aus, die von der `INPUT`-Anweisung gelesen werden kann. Dazu werden die einzelnen Ausdrücke durch Kommata getrennt und Zeichenketten in Anführungszeichen eingeschlossen.

Syntax:

```
WRITE [#n,] a1|s1$ [,a2|s2$..]
```

Siehe auch:

```
PRINT
```

WRITEPREFS *Anweisung*

Sichert diverse Voreinstellungen dauerhaft, so daß sie nach dem Ausschalten wieder zur Verfügung stehen.

Syntax:

```
WRITEPREFS
```

Siehe auch:

```
CALIBRATE, CONTRAST, KEYREPEAT, STACKSIZE
```

XPRINT *Anweisung*

Gibt Zeichenketten oder Zeichenkettenrepräsentationen von numerischen Ausdrücken in eine angegebene Dateinummer `n` aus. Ausdrücke, die in der Ausdrucksliste mit einem Semikolon getrennt werden, werden unmittelbar nacheinander ausgegeben, mit einem Komma getrennte Ausdrücke werden durch ein Tabulatorzeichen (ASCII-Code 9) getrennt. Die Ausgabe wird mit einem CR/LF (ASCII-Codes 13 und 10) abgeschlossen, falls am Ende der `PRINT`-Anweisung kein Semikolon oder Komma steht.
Wenn `n=0` ist, wird der Standard-Ausgabekanal verwendet.

Syntax:

```
XPRINT n[,[a1|s1$][;|,]..
```

Beispiel:

```
XPRINT outchan,"HALLO";CHR$(33)
XPRINT 2,100.0,20;
```

Siehe auch:

```
PRINT, PRINT USING, WRITE
```

Motorsteuerungs-Anweisungen

Konventionen:

a	numerischer Ausdruck mit Integer- oder Fixkommaergebnis
ax	numerischer Ausdruck, der die x-Koordinate angibt
ay	numerischer Ausdruck, der die y-Koordinate angibt
az	numerischer Ausdruck, der die z-Koordinate angibt
aw	numerischer Ausdruck, der einen Winkel angibt (in Grad) gemessen bezüglich der positiven X-Achse, negativ im Uhrzeigersinn, positiv gegen den Uhrzeigersinn
n	Integerzahl
na	Integerzahl, die die Achsnummer angibt (na=1..3)
ng	Integerzahl, die die Gruppennummer angibt (ng=1..8)
c	Einzelnes ASCII-codiertes Zeichen
s$	Zeichenkette (String)
[..]	optionaler Parameter
Syntax 1	zu verwendende Syntax, falls die verwendete Gruppe aus einer Achse besteht
Syntax 2	zu verwendende Syntax, falls die verwendete Gruppe aus zwei Achsen besteht
Syntax 3	zu verwendende Syntax, falls die verwendete Gruppe aus drei Achsen besteht

ACC *Anweisung*

Setzt die Beschleunigung (in Einheiten/s²) der Achsgruppe ng auf den Wert a. Die Anweisung kann auch während einer Fahrt aufgerufen werden, der neue Wert wird sofort berücksichtigt.

Syntax:

```
ACC ng,a
```

Siehe auch:

DEC, HDEC, VEL, TRAFO

ALENGTH *Funktion*

Gibt die aktuelle Bogenlängenposition der Gruppe ng auf einer Kontur zurück.

Syntax:

```
a=ALENGTH(ng)
```

Beispiel:

```
PRINT ALENGTH(1)
```

Siehe auch:

CONTOUR, LENGTH

AMPERROR *Anweisung*

Schaltet die Ereignisverfolgung der Achse na im Falle eines Endstufenfehlers ein oder aus. Dabei wird bei AMPERROR(na) HIGH ein logisches High am Fehlereingang als Fehler interpretiert, bei AMPERROR(na) LOW ein logisches Low.

Syntax:

```
AMPERROR(na) HIGH|LOW|OFF
```

Siehe auch:
MAXDIFF, MAXFORCE, MOTOR

APOS

Funktion

Gibt die aktuelle Position der n-ten Achse der Gruppe ng zurück. Bei Servomotoren wird der skalierte Wert des entsprechenden Inkrementalgebers ausgegeben (Istwert), nicht der Sollwert des Reglers (siehe TPOS).

Syntax:
a=APOS(ng,n)

Beispiel:
```
PRINT APOS(1,1),APOS(1,2),APOS(1,3)
```

Siehe auch:
CPOS, TP, TPOS

ARCA

Anweisung

Startet eine Kreisbogenfahrt um den Winkel aw von der aktuellen Position, um den in absoluten Koordinaten angegebenen Mittelpunkt ax,ay,az. Der Modusparameter nm gibt an, ob die Kreisfahrt in der X-Y-Ebene (nm=0), in der YZ-Ebene (nm=1) oder in der XZ-Ebene (nm=2) stattfinden soll.

Syntax 2:
```
ARCA ng,ax,ay,aw
```

Syntax 3:
```
ARCA ng,0,ax,ay,aw
ARCA ng,1,ay,az,aw
ARCA ng,2,ax,az,aw
```

Beispiel:
```
ARCA 1,100.0,50.0,180
```

Siehe auch:
ARCR

ARCR

Anweisung

Startet eine Kreisbogenfahrt um den Winkel aw von der aktuellen Position, um den in relativen Koordinaten angegebenen Mittelpunkt ax,ay,az. Der Modusparameter nm gibt an, ob die Kreisfahrt in der X-Y-Ebene (nm=0), in der YZ-Ebene (nm=1) oder in der XZ-Ebene (nm=2) stattfinden soll.

Syntax 2:
```
ARCR ng,ax,ay,aw
```

Syntax 3:
```
ARCR ng,0,ax,ay,aw
ARCR ng,1,ay,az,aw
ARCR ng,2,ax,az,aw
```

Beispiel:
```
ARCR 1,+50.0,-50.0,180
```

Siehe auch:
ARCA

AUTOVEL *Anweisung*

Schaltet die automatische Geschwindigkeitsreduzierung für Kreisbogenfahrten ein oder aus. Ist die Geschwindigkeitsreduzierung eingeschaltet, wird die Geschwindigkeit in Kreisbögen so reduziert, daß das Minimum von eingestellter Beschleunigung und Verzögerung nicht überschritten wird.

Syntax:

```
AUTOVEL [ng1[,ng2..]] ON|OFF
```

Beispiel:

```
AUTOVEL ON
```

Siehe auch:

```
DEC, ACC, MAXSEGMENTS, RSEGMENTS
```

AVEL *Funktion*

Gibt die aktuelle Verfahrgeschwindigkeit der Gruppe `ng` bzw. der Achse `n` der Gruppe `ng` zurück (`n=1`: X-Achse, `n=2`: Y-Achse, `n=3`: Z-Achse).

Syntax 1: Linearfahrt oder Kreisfahrt

```
a=AVEL(ng)
```

Syntax 2: Positionsfahrt

```
a=AVEL(ng,n)
```

Siehe auch:

```
VEL
```

BORDER *Anweisung*

Begrenzt den Verfahrbereich durch eine untere (`a1`) und eine obere Grenze (`a2`) pro Achse. Wenn die Grenze bei einer Achsbewegung erreicht wird, wird die betreffende Achse ohne Bremsrampe gestoppt.

Syntax 1:

```
BORDER ng,a1x,a2x
```

Syntax 2:

```
BORDER ng,a1x,a2x,a1y,a2y
```

Syntax 3:

```
BORDER ng,a1x,a2x,a1y,a2y,a1z,a2z
```

Beispiel:

```
BORDER 1,-200,+200,-130,+150
```

CODE *Funktion*

Gibt den zuletzt gesetzten Wert des nach der G-Code-Syntax gesetzten Codes zurück. Für den Code `sel` sind dabei folgende Werte erlaubt:
Fixkommaparameter: `A`, `F`, `I`, `J`, `K`, `R`, `S`, `W`, `X`, `Y`, `Z`
Integerparameter: `G`, `L`, `M`, `N`
Spezielle Parameter: `C`

Syntax:

```
a=CODE(sel)
```

Beispiel:

```
F100.5 G02
```

```
I+100 J-55.5
PRINT CODE(F),CODE(G),CODE(I),CODE(J)
```
Siehe auch:
```
ON CTRLCODE GOTO|GOSUB
```

CONTINUOUS *Anweisung*

Schaltet den kontinuierlichen Verfahrmodus für die Gruppen ng1, ng2, ... ein oder aus. Ist der kontinuierliche Modus eingeschaltet, wird die konstante Geschwindigkeit nach der Beschleunigungsphase solange beibehalten, wie Fahrbefehle (LINA, LINR, ARCA, ARCR) nachgereicht werden. Zuvor muß die Anweisung NOWAIT ON ausgeführt werden.

Syntax:
```
CONTINUOUS ng1[,ng2[,ng3..]] ON|OFF
```
Siehe auch:
```
NOWAIT
```

CONTOUR *Anweisung*

Schaltet den Konturmodus für die Gruppen ng1, ng2, ... ein oder aus. Eine Kontur besteht aus einem oder mehreren Linear- und Zirkularsegmenten, die nach CONTOUR ON mit LINA, LINR, ARCA und ARCR programmiert werden können. Ausgangspunkt ist die aktuelle Position, jedoch kann diese mit POSA bzw. POSR verschoben werden. Im Kontur-Eingabemodus finden keine Achsbewegungen der betroffen Gruppe statt. Nach Beendigung der Kontureingabe kann die Kontur mit MOVER oder MOVEA-Anweisungen abgefahren werden.

Syntax:
```
CONTOUR ng1[,ng2[,ng3..]] ON|OFF
```
Beispiel:
```
CONTOUR ON
LINR 1,0,100
ARCR 1,25,0,-180
CONTOUR OFF
```
Siehe auch:
```
ALENGTH, CTPOS, LENGTH, MOVEA, MOVER, NEWSEG,
SETCPOS
```

CONTSEG *Anweisung*

Bewirkt, daß beim späteren Abfahren einer Kontur abweichend vom gewählten Segment-Modus (SEGMODE ON) nach dem zuletzt eingegebenen Segment die Geschwindigkeit konstant bleibt z.B. wenn eine Linearfahrt tangential an einen Kreisbogen anschließt.

Syntax:
```
NEWSEG ng
```
Siehe auch:
```
CONTOUR, SEGMODE, CONTSEG
```

CPOS

Funktion

Gibt die Endposition der letzten Fahranweisung der `n`-ten Achse der Gruppe `ng` zurück.

Syntax:

```
a=CPOS(ng,n)
```

Beispiel:

```
NOWAIT ON
LINA 1,100,200,300
PRINT CPOS(1,1),CPOS(1,2),CPOS(1,3)
```

Siehe auch:

`APOS, TP, TPOS`

CTPOS

Funktion

Gibt die Position der `n`-ten Achse der Gruppe `ng` innerhalb einer Kontur bei der Bogenlänge `l` zurück.

Syntax:

```
a=CTPOS(ng,n,l)
```

Beispiel:

```
CONTOUR ON
LINR 1,100,150
CONTOUR OFF
PRINT CTPOS(1,1,50),CTPOS(1,2,50)
```

Siehe auch:

`CONTOUR, SETCPOS, LENGTH, ALENGTH`

CVEL

Funktion

Gibt die mit `VEL`, bzw. `POSVEL` programmierte maximale Verfahrgeschwindigkeit der Gruppe `ng` bzw. der Achse `n` der Gruppe `ng` zurück (`n=1`: X-Achse, `n=2`: Y-Achse, `n=3`: Z-Achse).

Syntax 1: `VEL`

```
a=CVEL(ng)
```

Syntax 2: `POSVEL`

```
a=CVEL(ng,n)
```

Siehe auch:

`AVEL, VEL, POSVEL`

DEC

Anweisung

Setzt die Bremsverzögerung (in Einheiten/s²) der Achsgruppe `ng` auf den Wert `a`. Die Anweisung kann auch während einer Fahrt aufgerufen werden, der neue Wert wird sofort berücksichtigt.

Syntax:

```
DEC ng,a
```

Beispiel:

```
DEC 1,200
```

Siehe auch:

`ACC, HDEC, VEL, TRAFO`

DEFAULTCODE *Anweisung*

Stellt das Verhalten beim Abschluß einer G-Code-Zeile ein. Mit der Anweisung `DEFAULTCODE ON` wird eine Zeile, in der nur Koordinaten, aber keine G-Anweisung auftauchen automatisch als `G01`-Anweisung (Linearfahrt) interpretiert.
Mit `DEFAULTCODE OFF` werden nach einer solchen Zeile nur die Code-Flags gesetzt, aber es wird keine Fahranweisung ausgeführt.

Syntax:
`DEFAULTCODE ON|OFF`

Siehe auch:
`G-Codes`

FMAX *Anweisung*

Setzt die Geschwindigkeit (in Einheiten/s), die durch den `F`-Code maximal eingestellt werden darf.

Syntax:
`FMAX a`

Beispiel:
`FMAX 200`

Siehe auch:
`G-Codes`

G-Codes *Anweisungen*

Siehe Anhang A.

GETV *Funktion*

Gibt des aktuellen Wert eines Achsparameters zurück.

Syntax:
`a=GETV(na,modus)`

Modus:

DEADBAND
Gibt den Toleranzbereich des PID-Reglers zurück.

DIFF
Gibt den Differentialanteil des PID-Reglers zurück.

FACTOR
Gibt den Skalierungsfaktor zwischen `input1` und `input2` beim PIDD-Modus zurück.

FEEDFWD
Wird noch nicht unterstützt.

FORCE
Gibt den aktuellen Reglerausgang zurück (Wert –100 bis +100).

IMAX
Gibt den Wert des maximalen Dauerstromes zurück.

INT
Gibt den Wert des Integralanteil des PID-Reglers zurück.

IPEAK
Gibt den Wert des Spitzenstromes zurück, falls `na` als

Servomotor konfiguriert ist.

ISTANDBY

Gibt den Wert des Ruhestroms zurück, falls `na` als Schrittmotor konfiguriert ist.

ITIME

Gibt den Wert der Spitzenstrom-Zeitbegrenzung zurück.

MAXFORCE

Gibt den maximal zulässigen Wert des Reglerausgangs zurück.

MAXDIFF

Gibt die maximal zulässige Abweichung des Reglersollwertes zum Istwert zurück.

MOTOR

Gibt den aktuellen Regelzustand zurück.

OFFSET

Gibt den Offsetwert des Reglerausgangs zurück.

POL

Gibt die Zählrichtung der Inkrementalgebereingänge zurück.

PROP

Gibt den Proportionalanteil des PID-Reglers zurück.

PWMOFFSET

Gibt den PWM-Offset des PWM-Ausgangs zurück.

PWMPOL

Gibt die PWM-Polarität des PWM-Ausgangs zurück.

RAPOS

Liest die aktuelle Zählerposition der Inkrementalgeberzähler.

RTPOS

Liest die aktuelle Sollposition der Achse.

SLFACTOR

Gibt den Skalierungsfaktor zwischen der Master und der Slaveachse zurück.

Siehe auch:

`SET`

GROUP

Anweisung

Faßt eine oder mehrere Achsen zu einer Gruppe zusammen. Nur innerhalb einer Gruppe kann linear bzw. zirkular interpoliert werden. Die Reihenfolge, in der die Achsen angegeben werden, bestimmt dabei, welche als X, Y oder Z-Achse bezeichnet werden kann. Falls nur die Gruppennummer angegeben wird, wird die entsprechende Gruppe aufgelöst.

Syntax:

```
GROUP ng,na1,[na2[,na3]]
GROUP ng
```

Siehe auch:

`HDEC, POSHDEC, STOP`

HALT

Anweisung

Die Bewegung der Gruppen `ng1`, `ng2`, ... wird beendet, wobei der Bremsverzögerungswert verwendet wird, der mit den

Befehlen HDEC bzw. POSHDEC gesetzt wurde.
Syntax:
HALT ng1,[ng2[,ng3..]]
Siehe auch:
HDEC, POSHDEC, STOP

HDEC

Anweisung
Setzt die Bremsverzögerung, die bei der HALT-Anweisung verwendet wird, auf den Wert a (in Einheiten/s²). Die Anweisung kann auch während einer Fahrt aufgerufen werden, der neue Wert wird sofort berücksichtigt.
Syntax:
HDEC ng,a
Beispiel:
HDEC 1,1000
Siehe auch:
ACC, DEC, VEL, TRAFO, HALT

INC

Funktion
Gibt den aktuellen Wert von Inkrementaleingang n zurück.
Syntax:
a=INC(n)
Beispiel:
PRINT INC(1)
Siehe auch:
INCFILTER

INCFILTER

Anweisung
Setzt einen Filter für den Inkrementalgebereingang n1. Der Filterparameter n2 gibt die Mindeständerung der Inkrementalgeberposition an. Durch den Filter kann in manchen Einsatzfällen die Regelstabilität auf Kosten der Genauigkeit erhöht werden.
Syntax:
INCFILTER n1,n2
Beispiel:
INCFILTER 1,1
Siehe auch:
INC

INDEX

Anweisung
Schaltet die Ereignisverfolgung der Indeximpulserkennung von Achse na ein oder aus. INDEX(na) RISING detektiert eine steigende Flanke am Indexeingang, INDEX(na) FALLING eine fallende Flanke. Die Indeximpulserkennung ist je nach Ausstattung nur bei einigen Steuerungen möglich.
Syntax:
INDEX(na) RISING|FALLING|OFF
Siehe auch:

```
ON xx GOTO|GOSUB
```

JUMP

Anweisung

Setzt an der Bahnsteuerung vorbei direkt eine neue Sollposition `n` für die Achse `na`. Der Einsatz dieser Anweisung wird nur für spezielle Anwendungen empfohlen. Es findet keine Überprüfung bezüglich maximaler Beschleunigung oder Geschwindigkeit statt, keine Verfahrbereichsüberwachung und auch keine Koordinatentransformation. Ein Einsatzfall ist beispielsweise das Provozieren einer Sprungantwort zur Einstellung der Regelparameter bei PID-Reglern. Die Positionen dürfen bei Schrittmotorendstufen nicht häufiger als alle 1.953 ms (512 Hz) und bei Servomotorendstufen nicht häufiger als alle 0.977 ms (1024 Hz) gesetzt werden.

Syntax:

```
JUMP na,n
```

Beispiel:

```
JUMP 1,50
```

LENGTH

Funktion

Gibt die gesamte Bogenlänge der Kontur der Gruppe `ng` zurück.

Syntax:

```
a=LENGTH(ng)
```

Beispiel:

```
PRINT LENGTH(1)
```

Siehe auch:

```
ALENGTH, CONTOUR
```

LINA

Anweisung

Startet eine Linearfahrt von der aktuellen Position aus zu den absoluten Koordinaten `ax,ay,az`.

Syntax 1:

```
LINA ng,ax
```

Syntax 2:

```
LINA ng,[ax][,[ay]]
```

Syntax 3:

```
LINA ng,[ax][,[ay]][,az]
```

Beispiel:

```
LINA 1,100.0,50.0,100.0
```

Siehe auch:

```
LINR
```

LINR

Anweisung

Startet eine Linearfahrt von der aktuellen Position aus zu den relativen Koordinaten `ax,ay,az`.

Syntax 1:

```
LINR ng,ax
```

Syntax 2:

```
LINR ng,[ax][,[ay]]
```
Syntax 3:
```
LINR ng,[ax][,[ay]][,az]
```
Beispiel:
```
LINR 1,-100.0,+50.0,-100.0
```
Siehe auch:
```
LINA
```

MAXDIFF *Anweisung*

Schaltet die Ereignisverfolgung der Schleppfehlerüberwachung von Achse `na` ein oder aus. Der Schleppfehler ist der Betrag der Differenz zwischen Soll- und Istwert des Reglers. Der maximale Betrag kann mit dem Befehl `SET na,MAXDIFF,a` eingestellt werden. Mit dem Parameter `AUTO` werden Regelung und Endstufe der Achse `na` bei Überschreitung des Schleppabstandes automatisch ausgeschaltet, mit dem Parameter `MANUAL` wird diese Automatik wieder ausgeschaltet.

Syntax:
```
MAXDIFF(na) ON|OFF
MAXDIFF(na) AUTO|MANUAL
```
Siehe auch:
```
AMPERROR, MAXFORCE, SET, ON xx GOTO|GOSUB
```

MAXFORCE *Anweisung*

Schaltet die Ereignisverfolgung der Reglerausgangsüberwachung von Achse `na` ein oder aus. Damit kann verhindert werden, daß der Reglerausgang seinen Maximalwert annimmt, denn dies bedeutet, daß der Regler die gewünschte Position nicht erreicht. Der maximale Betrag kann mit dem Befehl `SET na,MAXFORCE,a` eingestelt werden.

Syntax:
```
MAXFORCE(na) ON|OFF
```
Siehe auch:
```
AMPERROR, MAXDIFF, SET, ON xx GOTO|GOSUB
```

MAXSEGMENTS *Anweisung*

Setzt die Zahl der Segmente, die im `AUTOVEL`- und `CONTOUR`-Modus überprüft werden, um die maximale Geschwindigkeit zu berechnen.

Syntax:
```
MAXSEGMENTS ng,n
```
Siehe auch:
```
AUTOVEL, RSEGMENTS
```

MODE *Anweisung*

Stellt die Motor- und Endstufenparameter ein. Es gibt zwei Syntaxvarianten, eine für Schrittmotoren, eine für Servomotoren.

Syntax für Servomotoren:
`MODE na,PID,input,output`
Als Istwerteingang `input` kann folgendes angegeben werden:

`INC(n)`	Inkrementalgeber
`ADIN(n)`	Analoger Eingang (je nach Ausstattung)

Als Reglerausgang `output` ist möglich:

`PWMS(n)`	PWM-Ausgang mit Vorzeichen
`PWMD(n)`	Differentieller PWM-Ausgang
`DAOUT(n)`	Analoger Ausgang
`UDAOUT(n)`	Unipolarer analoger Ausgang mit Vorzeichen

Syntax für Servomotoren mit 2 Positionsgebern:
`MODE na,PIDD,input1,input2,output`
In diesem Modus wird der Proportional- und Integralanteil über einen Positionsgeber (`input2`, z.B. ein Linearmaßstab), der Differentialanteil über einen anderen Positionsgeber (`input1`, z.B. Rotationsgeber) berechnet. Der Skalierungsfaktor zwischen `input1` und `input2` wird mit der Anweisung `SET na,FACTOR,a` eingestellt.
Als Istwerteingänge `input1/2` kann folgendes angegeben werden:

`INC(n)`	Inkrementalgeber
`ADIN(n)`	Analoger Eingang (je nach Ausstattung)

Als Reglerausgang `output` ist möglich:

`PWMS(n)`	PWM-Ausgang mit Vorzeichen
`PWMD(n)`	Differentieller PWM-Ausgang
`DAOUT(n)`	Analoger Ausgang
`UDAOUT(n)`	Unipolarer analoger Ausgang mit Vorzeichen

Syntax für Schrittmotoren:
`MODE na,SM,output`
Als Ausgang `output` ist möglich:

`STEPDIR(n)`	Schritt/Richtungsausgang
`PATTERN(n)`	Patterngenerator (nur bei MC-Baureihe)

Die Motor- und Endstufenparameter können nur einmal nach dem Neustart eingestellt werden, eine nachträgliche Änderung ist nicht möglich.
Beispiel:

```
MODE 1,PID,INC(1),DAOUT(1)
MODE 2,PID,INC(2),PWMS(2)
MODE 3,SM,STEPDIR(1)
```

Siehe auch:
`SET`

MOTOR *Anweisung*

Schaltet die Endstufen sowie die Positionsregelung bei Servomotoren ein oder aus.
Syntax:

```
MOTOR [na1][,na2][,na3..] ON|OFF
```
Siehe auch:
```
SET, ON xx GOTO|GOSUB
```

MOVEA *Anweisung*

Startet eine Konturfahrt zur absoluten Bogenlängenposition `l`. Dabei wird die aktuelle Bodenlängenposition auf der Kontur zunächst mit einer Posititionsfahrt angefahren und von dort aus die eigentliche Konturfahrt begonnen.

Syntax:
```
MOVEA ng,l
```
Beispiel:
```
MOVEA 1,120.0
```
Siehe auch:
```
CONTOUR, MOVER
```

MOVER *Anweisung*

Startet eine Konturfahrt der relativen Bogenlängen `l`. Dabei wird die aktuelle Bodenlängenposition auf der Kontur zunächst mit einer Posititionsfahrt angefahren und von dort aus die eigentliche Konturfahrt begonnen.

Syntax:
```
MOVER ng,l
```
Beispiel:
```
MOVER 1,120.0
```
Siehe auch:
```
CONTOUR, MOVEA
```

MOVING *Funktion*

Gibt den Wert `TRUE` zurück, falls Gruppe `ng` noch in einer Fahbewegung ist, `FALSE`, wenn diese beendet ist.

Syntax:
```
a=MOVING(ng)
```
Siehe auch:
```
NOWAIT, WAIT
```

NEWCODE *Funktion*

Gibt den Wert `TRUE` zurück, falls mit dem G-Code `sel` neue Parameter übergeben wurden. Für den Code `sel` sind folgende Buchstaben erlaubt:
Fixkommaparameter: `A`, `F`, `I`, `J`, `K`, `R`, `S`, `W`, `X`, `Y`, `Z`
Integerparameter: `G`, `L`, `M`, `N`

Syntax:
```
a=NEWCODE(sel)
```
Siehe auch:
```
CODE, G-Codes, RESETCODEFLAGS
```

NEWSEG *Anweisung*

Bewirkt, daß beim späteren Abfahren einer Kontur abweichend vom gewählten Segment-Modus (SEGMODE OFF) nach dem zuletzt eingegebenen Segment die Bewegung angehalten wird, z.B. wenn die Kontur einen Winkel beinhaltet.

Syntax:

```
NEWSEG ng
```

Siehe auch:

CONTOUR, SEGMODE, CONTSEG

NOTAUS *Anweisung*

Schaltet die Ereignisverfolgung im Falle eines Notaus ein oder aus, stellt die Art der Notausverwaltung ein oder löst einen Notaus aus.
Im INTERN-Modus kann ein eingebauter Schütz aktiv geschaltet werden, d.h. die START-Anweisung oder ein Druck auf die **START**-Taste schaltet ihn an, und die NOTAUS-Anweisung oder ein Druck auf die **STOP**-Taste schaltet ihn aus. Im EXTERN-Modus wird ein externer Notauskreis überwacht, dabei können die **START**- und die **STOP**-Taste zusätzlich zu den vorhandenen Tasten verwendet werden. Der MIXED-Modus erlaubt im Gegensatz zum INTERN-Modus die Verwendung eines externen **NOTAUS**-Tasters.

Syntax:

```
NOTAUS INTERN|EXTERN|MIXED  ; Verwaltung
NOTAUS                      ; NOTAUS auslösen
NOTAUS ON|OFF           ; Ereignisverfolgung
```

Siehe auch:

KEYTIME, START

NOWAIT *Anweisung*

Stellt das Verhalten des Interpreters nach einer Positionier- oder Fahranweisung für die Gruppen ng1, ng2.. ein. Ist NOWAIT ON eingestellt, nimmt der Interpreter sofort wieder Befehle entgegen, während die Bewegung läuft, bei NOWAIT OFF wird gewartet, bis die Bewegung beendet ist.

Syntax:

```
NOWAIT [ng1][,ng2][,ng3..] ON|OFF
```

Siehe auch:

MOVING, WAIT

ON *Anweisung*

Stellt die Programmzeile n ein, zu der bei dem Ereignis event gesprungen werden soll. Bei den Ereignissen AMPERROR, MAXDIFF und MAXFORCE darf nur GOSUB verwendet werden. Ist eine GOTO-Verzweigung erwünscht, verwendet man eine GOSUB-POPSTACK-Kombination. Vor dem Befehl POPSTACK (oder CLEARSTACK) muß die Ereignisursache beseitigt werden (z.B. mit MOTOR OFF). Es sind folgende Ereignisse möglich:

AMPERROR(na)
Die Endstufenfehlerleitung hat den mit `AMPERROR(na)` angegebenen Pegel angenommen.

CODE(sel)
Dieses Ereignis tritt auf, falls ein G-Code mit dem Buchstaben `sel` (`G`, `E` (Zeilenende), `D`, `M`, `T`) eingegeben wurde. Weitere Hinweise in Anhang A: G-Codes.

INDEX(na)
Am Indexeingang `n` ist der mit `INDEX(na)` angegebene Signalwechsel aufgetreten.

MAXDIFF(na)
Die mit `SET na,MAXDIFF,a` angegebene maximale Regelabweichung wurde überschritten.

MAXFORCE(na)
Der mit `SET na,MAXFORCE,a` angegebene maximale Reglerausgangswert wurde überschritten.

NOTAUS
Die Notaus-Schütz Fühlerleitung S+ hat ihren Zustand von +24V auf 0V gewechselt.

START
Die Notaus-Schütz Fühlerleitung S+ hat ihren Zustand von 0V auf +24V gewechselt.

TOOLOFF
Dieses Ereignis tritt auf, falls auf einen Fahr-G-Code (`G01`, `G02`, `G03`) ein Positionier-G-Code (`G00`) folgt.

TOOLON
Dieses Ereignis tritt auf, falls auf einen Positionier-G-Code (`G00`) ein Fahr-G-Code (`G01`, `G02`, `G03`) folgt.

Syntax:
```
ON event GOTO|GOSUB n
```
Siehe auch:
`AMPERROR, G-Codes, INDEX, MAXDIFF, MAXFORCE, SET, NOTAUS, START, TOOLOFF, TOOLON`

POSA *Anweisung*

Beginnt eine Positionsfahrt der zur Gruppe `ng` gehörenden Achsen zu den angegebenen absoluten Koordinaten.

Syntax:
```
POSA ng,[ax][,[ay][,az]]
```
Beispiel:
```
POSA 1,100.5,,25
```
Siehe auch:
`POSACC, POSDEC, POSHDEC, POSR, POSVEL`

POSACC *Anweisung*

Setzt die Beschleunigung (in Einheiten/s²) der Achsen der Achsgruppe `ng`. Die Anweisung kann auch während einer Fahrt aufgerufen werden, der neue Wert wird sofort berücksichtigt.

Syntax 1:
```
POSACC, ng,ax
```
Syntax 2:
```
POSACC ng,[ax][,[ay]
```
Syntax 3:
```
POSACC ng,[ax][,[ay][,az]]
```
Beispiel:
```
POSACC 1,1000,1000,100
```
Siehe auch:
```
POSA, POSDEC, POSHDEC, POSR, POSVEL
```

POSDEC *Anweisung*

Setzt die Bremsverzögerung (in Einheiten/s²) der Achsen der Achsgruppe `ng`. Die Anweisung kann auch während einer Fahrt aufgerufen werden, der neue Wert wird sofort berücksichtigt.

Syntax 1:
```
POSDEC, ng,ax
```
Syntax 2:
```
POSDEC ng,[ax][,[ay]
```
Syntax 3:
```
POSDEC ng,[ax][,[ay][,az]]
```
Beispiel:
```
POSDEC 1,1000,1000,100
```
Siehe auch:
```
POSA, POSACC, POSHDEC, POSR, POSVEL
```

POSHDEC *Anweisung*

Setzt die Bremsverzögerung (in Einheiten/s²) der Achsen der Achsgruppe `ng`. Dieser Verzögerungswert wird eingesetzt, falls die `HALT`-Anweisung verwendet wird. Die Anweisung kann auch während einer Fahrt aufgerufen werden, der neue Wert wird sofort berücksichtigt.

Syntax 1:
```
POSHDEC, ng,ax
```
Syntax 2:
```
POSHDEC ng,[ax][,[ay]
```
Syntax 3:
```
POSHDEC ng,[ax][,[ay][,az]]
```
Beispiel:
```
POSHDEC 1,5000,5000,2000
```
Siehe auch:
```
POSA, POSACC, POSDEC, POSR, POSVEL
```

POSR *Anweisung*

Beginnt eine Positionsfahrt der zur Gruppe `ng` gehörenden Achsen zu den angegebenen relativen Koordinaten.

Syntax:
`POSA ng,[ax][,[ay][,az]]`
Beispiel:
`POSA 1,100.5,,25`
Siehe auch:
`POSA, POSACC, POSDEC, POSHDEC, POSVEL`

POSVEL

Anweisung
Setzt die Positioniergeschwindigkeit (in Einheiten/s) der Achsen der Achsgruppe `ng`. Die Anweisung kann auch während einer Fahrt aufgerufen werden, der neue Wert wird sofort berücksichtigt.
Syntax 1:
`POSVEL ng,ax`
Syntax 2:
`POSVEL ng,[ax][,[ay]`
Syntax 3:
`POSVEL ng,[ax][,[ay][,az]]`
Beispiel:
`POSVEL 1,100,100,20`
Siehe auch:
`POSACC, POSDEC, POSHDEC`

POWERON

Funktion
Gibt den Status des Notaus-Kreises zurück. Der Rückgabewert ist `TRUE`, falls die **START**-LED grün leuchtet (d.h. bei korrekter Verschaltung ist die Versorgungsspannung der Endstufen eingeschaltet), und `FALSE`, falls die **START**-LED rot leuchtet (Endstufen-Versorgung aus).
Syntax:
`a=POWERON`
Beispiel:
`PRINT POWERON`
Siehe auch:
`START, NOTAUS`

RESETCODEFLAGS

Anweisung
Setzt die Flags zurück, die durch G-Code-Befehle gesetzt wurden.
Syntax:
`RESETCODEFLAGS`
Siehe auch:
`G-Codes, CODE, NEWCODE`

RLENGTH

Anweisung
Gibt die verbleibende Restlänge der im `CONTINUOUS`-Modus zusammengefügten Segmente der Gruppe `ng` zurück.
Syntax:

```
a=RLENGTH(ng)
```
Siehe auch:
```
CONTINUOUS, RSEGMENTS
```

RSEGMENTS *Anweisung*

Gibt die Anzahl der im `CONTINUOUS`-Modus zusammengefügten Segmente der Gruppe `ng` zurück.

Syntax:
```
a=RSEGMENTS(ng)
```
Siehe auch:
```
AUTOVEL, CONTINUOUS, MAXSEGMENTS, RLENGTH
```

SCOPE *Anweisung*

Legt alle 0.977 ms (1024 Hz) die aktuelle Position der Achse `na` im Array `array(0)` ab, bis das Array gefüllt ist. Die Position ist die Position des Inkrementalgebers der Achse `na` (entsprechend `GETV(na,RAPOS)`) ohne Transformation. Das Array kann vom Typ Integer oder Char sein. Der Typ Char spart Speicherplatz, es tritt jedoch häufiger ein Überlauf auf, der durch ein Programm nachträglich korrigiert werden muß.

Syntax:
```
SCOPE na,array(0)
```
Beispiel:
```
DIM testarray(1000) AS INTEGER
SCOPE 1,testarray(0): SLEEP 50: JUMP 1,100
SLEEP 950
FOR i=1 to 1000: PRINT testarray(i): NEXT
```
Siehe auch:
```
JUMP
```

SEGMODE *Anweisung*

Schaltet den Segment-Modus innerhalb einer Kontureingabe ein oder aus. Ist der Segment-Modus eingeschaltet, wird zwischen jedem Kontursegment angehalten. Ist der Segment-Modus ausgeschaltet, wird mit konstanter Geschwindigkeit über die Schnittpunkte gefahren. Dies ist nur bei sehr geringen Winkelabweichungen zu empfehlen. Der Segment-Modus kann für einzelne Segmente mit dem `CONTSEG` bzw. `NEWSEG`-Befehl umgeschaltet werden.

Syntax:
```
SEGMODE ng1[,ng2[,ng3..]] ON|OFF
```
Siehe auch:
```
ALENGTH, CONTSEG, CTPOS, LENGTH, MOVEA, MOVER,
NEWSEG, SETCPOS
```

SET *Anweisung*

Mit der SET-Anweisung können unterschiedliche Parameter einer Achse gesetzt werden:

Syntax:

SET na,modus,a

Modus:

DEADBAND

Setzt das Toleranzband des PID-Reglers. Der Integer-Parameter a gibt an, ab welcher Regelabweichung (in Quadcounts) die Regelung einsetzt.

DIFF

Setzt den Differentialanteil des PID-Reglers. Der Parameter a kann Werte zwischen 0.0 und 32767.0 annehmen. Hinterkommastellen sind zulässig.

FACTOR

Setzt den Skalierungsfaktor zwischen input1 und input2 beim PIDD-Modus. FACTOR=Res(input2)/Res(input1), Res(n) ist die Auflösung des betreffenden Eingangs in Inkremente/Längeneinheit.

FEEDFWD

Wird noch nicht unterstützt.

IMAX

Setzt den Wert des maximalen Dauerstromes. a=0 ist der Minimal, a=100 der Maximalstrom (abhängig von der eingesetzten Endstufe). Bei den Steuerungen CO2200 und CO4300 wird ein eine entsprechende Steuerspannung zwischen 0 und 5V am Ausgang DA-A ausgegeben.

INT

Setzt den Integralanteil des PID-Reglers. Der Parameter a kann Werte zwischen 0.0 und 32767.0 annehmen. Hinterkommastellen sind zulässig.

IPEAK

Bei der Endstufe PW2000 wird mit diesem Parameter der maximale Motorspitzenstrom eingestellt. Bei den Steuerungen CO2200 und CO4300 wird ein eine entsprechende Steuerspannung zwischen 0 und 5V am Ausgang DA-B ausgegeben. Die Funktionsweise dieses Parameters ist identisch mit der des Parameters ISTANDBY.

ISTANDBY

Setzt den Wert des Ruhestroms, falls na als Schrittmotor konfiguriert ist. a=0 ist der Minimalstrom, a=100 der Maximalstrom (abhängig von der eingesetzten Endstufe). Bei der Endstufe PW2000 wird mit diesem Parameter der maximale Motorspitzenstrom eingestellt. Bei den Steuerungen CO2200 und CO4300 wird ein eine entsprechende Steuerspannung zwischen 0 und 5V am Ausgang DA-B ausgegeben.

ITIME

Bei Steuerungen mit integrierten stromüberwachten Servoendstufen (CO5500) wird mit diesem Parameter die Dauer der Überschreitung des Maximalstromes eingestellt. Der Spitzenstrom ist dabei auf den mit IPEAK eingestellten Wert begrenzt. Das zeitliche Integral über die Differenz zwischen

aktuellem Strom und dem Maximalstrom ist dabei proportional zum eingestellten Wert (0..100).

MAXFORCE

Setzt den maximal zulässigen Wert des Reglerausgangs. Es sind Werte zwischen 0 und 100 zulässig.

MAXDIFF

Setzt die maximal zulässige Abweichung des Reglersollwertes zum Istwert. Die Angabe erfolgt in Quadcounts im Bereich von 0 bis 32767.

MOTOR

Schaltet die Endstufe der Achse `na` ein oder aus. Der Parameter `a` kann `ON` oder `OFF` lauten.

OFFSET

Addiert einen Offsetwert auf den Reglerausgang. Dies kann z.B. zur Schwerkraftkompensation oder zur Korrektur von Endstufenoffsets eingesetzt werden. Es sind Werte zwischen –100 und +100 zulässig. Die Offsetstreuung der D/A-Ausgänge sollte mit `CALIBRATE` korrigiert weden.

PHASEA, PHASEB, PHASEC, PHASED

Setzt die 16 Bit breiten Phasenausgangsregister der Phasenausgänge A,B,C und D. Achse `na` muß als Schrittmotor konfiguriert sein. Diese Parameter sind nur bei der MC-Baureihe möglich.

POL

Setzt die Zählrichtung der Inkrementalgebereingänge. Der Parameter `a` kann `TRUE` oder `FALSE` lauten.

PROP

Setzt den Proportionalanteil des PID-Reglers. Der Parameter `a` kann Werte zwischen 0.0 und 32767.0 annehmen. Hinterkommastellen sind zulässig.

PWMOFFSET

Setzt den Null-Offset des PWM-Ausgangs. Dies dient dazu, eventuelles Offset-Verhalten von PWM-Endstufen zu kompensieren. Parameter `a` kann Werte zwischen -100 und +100 annehmen. Hinterkommastellen sind zulässig.

PWMPOL

Setzt die Polarität des PWM-Ausgangs. Parameter `a` kann `TRUE` oder `FALSE` sein.

RAPOS

Setzt die aktuelle Zählerposition der Inkrementalgeberzähler. Dieser Befehl sollte nur zu Diagnosezwecken verwendet werden.

RTPOS

Setzt die aktuelle Sollposition der Achse. Dieser Befehl sollte nur zu Diagnosezwecken verwendet werden.

SLFACTOR

Setzt den Skalierungsfaktor zwischen der Master- und der Slaveachse. Slaveposition=`SLFACTOR`*Masterposition. Dieser Befehl sollte nur verwendet werden, wenn zuvor die Positionen

der beteiligten Achsen mit SETPOS auf Null gesetzt wurden.

WAVEFORM1

Setzt die Phasenstromfunktion der ersten Phase bei Steuerungen mit Mikroschrittendstufen (CO6100 und CO6500). Parameter a ist ein eindimensionales Array der Länge 256 vom Typ FIX. Die Arraywerte dürfen zwischen –1 und +1 liegen. Bei der Version CO6150 müssen die Absolutwerte programmiert werden (zwischen 0 und 1), die Polarität wird automatisch so eingestellt, daß der Arraybereich von 1 bis 128 positive Ströme ergibt und der Bereich von 129 bis 256 negative (geeignet für ABS(SIN(x))).

WAVEFORM2

Setzt die Phasenstromfunktion der zweiten Phase bei Steuerungen mit Mikroschrittendstufen (CO6100 und CO6500). Parameter a ist ein eindimensionales Array der Länge 256 vom Typ FIX. Die Arraywerte dürfen zwischen –1 und +1 liegen. Bei der Version CO6150 müssen die Absolutwerte programmiert werden (zwischen 0 und 1), die Polarität wird automatisch so eingestellt, daß der Arraybereich von 1 bis 64 und von 193 bis 256 positive Ströme ergibt und der Bereich von 65 bis 192 negative (geeignet für ABS(COS(x))).

Siehe auch:
GETV

SETPOS

Anweisung

Setzt die aktuelle Position (Soll- und Istposition) der Gruppe ng.

Syntax 1:
```
SETPOS ng,ax
```
Syntax 2:
```
SETPOS ng,ax,ay
```
Syntax 3:
```
SETPOS ng,ax,ay,az
```
Siehe auch:
APOS, TPOS, CPOS, TP, TRAFO

SETCPOS

Anweisung

Setzt die aktuelle Bogenlänge der Gruppe ng innerhalb einer Kontur auf den Position l.

Syntax:
```
SETPOS ng,l
```
Siehe auch:
CONTOUR, CTPOS, MOVEA, MOVER

SLAVE

Anweisung

Kopplung einer Achse an eine andere. In Abhängigkeit vom Parameter APOS bzw TPOS wird für Achse na1 die Istposition (APOS) oder die Sollposition (TPOS) von Achse na2

übernommen. Falls nur der Parameter na1 angegeben wird, wird die Kopplung aufgehoben.

Syntax:

```
SLAVE na1,na2,APOS|TPOS
SLAVE na1
```

Beispiel:

```
SLAVE 3,2,APOS
```

START *Anweisung*

Dieser Befehl bewirkt ohne Parameter dasselbe wie das Drücken der **START**-Taste. Das EIN-Relais wird für eine mit KEYREPEAT eingestellte Zeit eingeschaltet, wenn dies zuvor mit dem Befehl START ENABLE erlaubt wurde. Mit den Parametern ON und OFF wird die Ereignisverfolgung ein- oder ausgeschaltet.

Syntax:

```
START [ON|OFF|ENABLE|DISABLE]
```

Siehe auch:

KEYREPEAT, NOTAUS

STATUS *Funktion*

Gibt den aktuellen Status von AMPERROR, MAXDIFF und MAXFORCE zurück. Bei MAXDIFF und MAXFORCE bedeutet die Rückgabe TRUE, daß die mit SET eingestellten Grenzen überschritten werden. MOTOR als Parameter gibt den Status der Regelung zurück (TRUE, wenn Regelung eingeschaltet).

Syntax:

```
n=STATUS(na,modus)
```

Beispiel:

```
PRINT STATUS(1,MAXDIFF)
```

Siehe auch:

SET

STOP *Anweisung*

Die Bewegung der Gruppen ng1, ng2,... wird sofort beendet, ohne daß eine Verzögerungsrampe verwendet wird.

Syntax:

```
STOP ng1,[ng2[,ng3..]]
```

Siehe auch:

HALT

TOOL *Anweisung*

Schaltet die Ereignisverfolgung für TOOLON und TOOLOFF-Ereignisse ein oder aus. Ein TOOLON-Ereignis wird ausgelöst, falls auf eine G00-Positionieranweisung eine G01/G02/G03-Fahranweisung folgt. Ein TOOLOFF-Ereignis wird ausgelöst, falls auf eine Fahranweisung eine Positionieranweisung folgt.

Syntax:

```
TOOL ON|OFF
```

Siehe auch:
`ON .. GOTO|GOSUB`

TP

Anweisung

Gibt die aktuellen Positionen aller Achsen in allen definierten Gruppe aus. Dabei werden die Achsen in ihrer Definitionsreihenfolge ausgegeben, beim Wechsel zur nächsten Gruppe wird eine neue Zeile begonnen.

Syntax:
`TP`

Siehe auch:
`APOS, CPOS, GROUP, TPOS`

TPOS

Funktion

Gibt die aktuelle Sollposition der `n`-ten Achse der Gruppe `ng` zurück. Bei Servomotoren wird nicht der Wert des entsprechenden Inkrementalgebers ausgegeben (Istwert), sondern der von der Bahnsteuerung berechnete Sollwert.

Syntax:
`a=TPOS(ng,n)`

Beispiel:
```
PRINT TPOS(1,1),TPOS(1,2),TPOS(1,3)
```

Siehe auch:
`APOS, CPOS, TP`

TRACKING

Anweisung

Schaltet den Tracking-Modus für die Gruppen `ng1`, `ng2`,... ein oder aus. Im Tracking-Modus werden Positionieranweisungen (`POSA`, `POSR`) vorzeitig beendet, falls mit einer neuen Positionieranweisung neue Koordinaten vorgegeben werden. Falls eine Richtungsänderung nötig ist, wird mit der entsprechenden Verzögerung gebremst und dann wieder beschleunigt.
Über mehrere Achsen interpolierende Fahranweisungen (`ARCA`, `ARCR`, `LINA`, `LINR`) sind nicht beeinflußt.

Syntax:
`TRACKING ng1,[ng2[,ng3..]] ON|OFF`

Siehe auch:
`POSA, POSR`

TRAFO

Anweisung

Setzt die Parameter für die Koodinatentransformationsmatrix der Gruppe `ng`.
Transformation der 1er-Gruppe:
```
x'= x*a
```

Transformation der 2er-Gruppe:
```
x'= x*ax1 + y*ay1
y'= x*ax2 + y*ay2
```

Transformation der 3er-Gruppe:

```
x'= x*ax1 + y*ay1
y'= x*ax2 + y*ay2
z'= z*az
```

Syntax 1:

```
TRAFO ng,a
```

Syntax 2:

```
TRAFO ng,ax1,ay1,ax2,ay2
```

Syntax 3:

```
TRAFO ng,ax1,ay1,ax2,ay2,az
```

VEL

Anweisung

Setzt die Geschwindigkeit (in Einheiten/s) der Achsgruppe `ng` auf den Wert `a`. Die Anweisung kann auch während einer Fahrt aufgerufen werden, der neue Wert wird sofort berücksichtigt.

Syntax:

```
VEL ng,a
```

Siehe auch:

```
ACC, DEC, HDEC, TRAFO
```

WAIT

Anweisung

Wartet, bis die angegebenen Gruppen ihre Fahrbewegungen beendet haben.

Syntax:

```
WAIT ng1[,ng2[,ng3..]]
```

Siehe auch:

```
NOWAIT, MOVING
```

3. Anhang A: G-Codes

G-Codes sind CNC-Programmieranweisungen, die nach einer bestimmten Syntax aufgebaut sind. Sie sind teilweise in dieser Steuerung implementiert, nicht implementierte Befehle können aber durch BASIC-Anweisungen emuliert werden. Ein G-Code besteht aus einem Buchstaben und einer Zahl, wobei für gewisse Buchstaben nur Integerzahlen (n), für andere auch Fixkommazahlen (f) erlaubt sind.
G-Codes werden mit den als Gruppe 1 (siehe GROUP-Anweisung) definierten Achsen ausgeführt. Gruppe 1 kann aus 2 oder 3 Achsen bestehen.
Pro Zeile darf nur ein G-Code des Typs G00, G01, G02, G03 oder G5n stehen. Es ist nicht möglich, Variablen als Parameter zu übergeben. In einer Zeile darf kein weiterer Befehl stehen, auch nicht mit : abgetrennt.

Siehe auch:
ON CODE(sel) GOTO|GOSUB, FMAX, CODE(sel), RESETCODEFLAGS, NEWCODE(sel), DEFAULTCODE, ON TOOLON GOTO|GOSUB, ON TOOLOFF GOTO|GOSUB, TOOL

Standard-G-Codes

Ff

Anweisung

Setzt die Verfahrgeschwindigkeit für die G01, G02 und G03-Anweisungen, ähnlich der VEL-Anweisung. Mit der Anweisung FMAX kann ein Maximalwert angegeben werden, damit durch die G-Code-Programmierung keine Maschinengrenzwerte überschritten werden können.

Syntax:
Ff

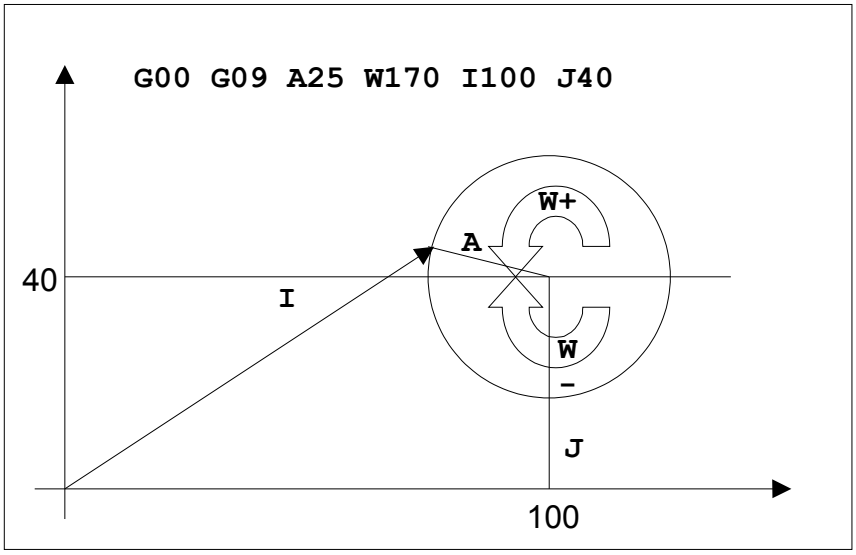
Beispiel:
```
F50
G02 X20 Y50 F20
```

Siehe auch:
FMAX, VEL

G00

Anweisung

Eilfahrt zu den angegebenen Koordinaten. Dabei werden die mit POSVEL, POSACC und POSDEC gesetzten Werte verwendet. Es werden nur die angegebenen Achsen verfahren, die anderen bleiben unverändert. Es sind relative Koordinaten (G91) und absolute Koordinaten (G90 oder keine Angabe) möglich, ebenso kartesische Koordinaten oder Polarkoordinaten (G09). Winkel werden in Grad angegeben und beziehen sich auf die positive X-Achse, wobei Winkel im Uhrzeigersinn negativ, Winkel gegen den Uhrzeigersinn positiv sind. Falls Achsgruppe 1 aus 3 Achsen besteht, entscheiden die angegebenen Parameter, in welcher Ebene die Polarkoordinaten berechnet werden, d.h. es dürfen nur jeweils 2 der 3 Koordinaten angegeben werden (z.B. I und J , J und K, K und I).



Bezeichungen:

`Xf`	X-Koordinate
`Yf`	Y-Koordinate
`Zf`	Z-Koordinate
`Af`	Radius
`Wf`	Winkel
`If`	X-Koordinate des Mittelpunktes bei Polarkoordinaten oder Kreisfahrt
`Jf`	Y-Koordinate des Mittelpunktes bei Polarkoordinaten oder Kreisfahrt
`Kf`	Z-Koordinate des Mittelpunktes bei Polarkoordinaten oder Kreisfahrt
`[...]`	Parameter kann weggelassen werden

absolute kartesische Koordinaten:
Die Koordinaten `X`, `Y` und `Z` geben die absolute Zielposition an.
Syntax:
```
G00 [G90] [Xf] [Yf] [Zf]
```
Beispiel:
```
G00 X100 Z-50
G00 Y-14.5
G00 G90 X10.5
```

relative kartesische Koordinaten:
`X`, `Y` und `Z` geben die Koordinaten der Zielposition relativ zur letzten Position an.
Syntax:
```
G00 G91 [Xf] [Yf] [Zf]
```
Beispiel:
```
G00 G91 X100 Z-50
G00 G91 Y-14.5
```

Polarkoordinaten, absoluter Mittelpunkt, absoluter Winkel:
I, J und K geben die absoluten Koordinaten des Mittelpunktes (bzw. des Poles) an, der Winkel ist in Grad bezüglich der positiven X-Achse angeben.
Syntax:
```
G00 G09 [M72] Af Wf [G90] If Jf     ' XY-Eb.
G00 G09 [M72] Af Wf [G90] Jf Kf     ' YZ-Eb.
G00 G09 [M72] Af Wf [G90] Kf If     ' ZX-Eb.
```
Beispiel:
```
G00 G09 A50 W75 I50 J50
G00 G09 M72 A25.5 W33.34 I+20 K-10
```

Polarkoordinaten, relativer Mittelpunkt, absoluter Winkel:
I, J und K geben die Koordinaten des Mittelpunktes (bzw. des Poles) relativ zur letzten Position an, der Winkel ist in Grad bezüglich der positiven X-Achse angeben.
Syntax:
```
G00 G09 [M72] Af Wf G91 If Jf       ' XY-Eb.
G00 G09 [M72] Af Wf G91 Jf Kf       ' YZ-Eb.
G00 G09 [M72] Af Wf G91 Kf If       ' ZX-Eb.
```
Beispiel:
```
G00 G09 A50 W75 G91 I10 J10
G00 G09 M72 A25.5 W33.34 G91 I-20.3 J0
```

Polarkoordinaten, absoluter Mittelpunkt, relativer Winkel:
I, J und K geben die absoluten Koordinaten des Mittelpunktes (bzw. des Poles) an, der Winkel ist relativ (im Kettenmaß) zum letzten verwendeten Winkel angegeben.
Syntax:
```
G00 G09 M71 Af Wf [G90] If Jf       ' XY-Eb.
G00 G09 M71 Af Wf [G90] Jf Kf       ' XZ-Eb.
G00 G09 M71 Af Wf [G90] Kf If       ' ZX-Eb.
```
Beispiel:
```
G00 G09 M71 A50 W15 I10 J10
G00 G09 M71 A25.5 W33.34 G90 I-20.3 J0
```

Polarkoordinaten, relativer Mittelpunkt, relativer Winkel:
I, J und K geben die Koordinaten des Mittelpunktes (bzw. des Poles) relativ zur letzten Position an, der Winkel ist relativ (im Kettenmaß) zum letzten verwendeten Winkel angegeben.
Syntax:
```
G00 G09 M71 Af Wf G91 If Jf         ' XY-Eb.
G00 G09 M71 Af Wf G91 Jf Kf         ' YZ-Eb.
G00 G09 M71 Af Wf G91 Kf If         ' ZX-Eb.
```
Beispiel:
```
G00 G09 M71 A50 W15 G91 I10 J10
G00 G09 M71 A25.5 W33.34 G91 I-20.3 J0
```
Siehe auch:
POSACC, POSDEC, POSVEL, G01

G01

Anweisung

Linearfahrt zu den angegebenen Koordinaten. Dabei werden für Beschleunigung und Verzögerung die mit ACC und DEC gesetzten Werte verwendet, die Geschwindigkeit entspricht dem letzten gesetzten F-Code. Für den F-Code muß mit der Anweisung FMAX zunächst ein Maximalwert gesetzt werden. Es werden nur die angegebenen Achsen verfahren, die anderen bleiben unverändert. Die Koordinatenangaben und die Syntax sind die gleichen wie bei der G00-Anweisung.

Siehe auch:

ACC, DEC, VEL, FMAX, Ff, G00

G02

Anweisung

Kreisfahrt im Uhrzeigersinn (negative Winkelrichtung) zu den angegebenen Koordinaten. Dabei werden für Beschleunigung und Verzögerung die mit ACC und DEC gesetzten Werte verwendet, die Geschwindigkeit entspricht dem letzten gesetzten F-Code. Falls Achsgruppe 1 aus 3 Achsen besteht, entscheiden die angegebenen Parameter, in welcher Ebene die Kreisfahrt stattfindet, d.h. es dürfen nur jeweils 2 der 3 Koordinaten angegeben werden (z.B. I und J, J und K, K und I, bzw. X und Y, Y und Z, Z und X).

Bezeichungen:

Xf	X-Koordinate
Yf	Y-Koordinate
Zf	Z-Koordinate
Af	Radius
Wf	Winkel
If	X-Koordinate des Mittelpunktes bei Polarkoordinaten oder Kreisfahrt
Jf	Y-Koordinate des Mittelpunktes bei Polarkoordinaten oder Kreisfahrt
Kf	Z-Koordinate des Mittelpunktes bei Polarkoordinaten oder Kreisfahrt
[...]	Parameter kann weggelassen werden

Absoluter Mittelpunkt, Vollkreis:

Die Koordinaten I, J und K geben den Kreismittelpunkt in absoluten Koordinaten an.

Syntax:

```
G02 [G90] If Jf                 ' XY-Ebene
G02 [G90] Jf Kf                 ' YZ-Ebene
G02 [G90] Kf If                 ' ZX-Ebene
```

Beispiel:

```
G02 I-50 J50
G02 G90 I22.2 J22.2
```

Relativer Mittelpunkt, Vollkreis:
Die Koordinaten I, J und K geben den Kreismittelpunkt relativ zur letzten Position an.
Syntax:
```
G02 G91 If Jf                    ' XY-Ebene
G02 G91 Jf Kf                    ' YZ-Ebene
G02 G91 Kf If                    ' ZX-Ebene
```
Beispiel:
```
G02 G91 I-50 J50
G02 G91 I22.2 J22.2
```

Kartesische Koordinaten, absoluter Mittelpunkt:
Die Koordinaten X, Y und Z geben den Zielpunkt der Kreisfahrt in absoluten Koordinaten an. Die Koordinaten I, J und K geben den Kreismittelpunkt in absoluten Koordinaten an.
Syntax:
```
G02 Xf Yf [G90] If Jf            ' XY-Ebene
G02 Yf Zf [G90] Jf Kf            ' YZ-Ebene
G02 Zf Xf [G90] Kf If            ' ZX-Ebene
```
Beispiel:
```
G02 X10 Y10 I50 J50
G02 X-30 Y20 G90 I22.2 J22.2
```

Kartesische Koordinaten, relativer Mittelpunkt:
Die Koordinaten X, Y und Z geben den Zielpunkt der Kreisfahrt in absoluten Koordinaten an. Die Koordinaten I und J geben den Kreismittelpunkt relativ zur letzten Position an.
Syntax:
```
G02 Xf Yf G91 If Jf              ' XY-Ebene
G02 Yf Zf G91 Jf Kf              ' YZ-Ebene
G02 Zf Xf G91 Kf If              ' ZX-Ebene
```
Beispiel:
```
G02 X10 Y10 G91 I-40 J50
G02 X-30 Y20 G91 I22.2 J22.2
```

Polarkoordinaten, absoluter Mittelpunkt, absoluter Winkel:
Der Winkel W ist der Endwinkel der Kreisfahrt bezüglich der positiven X-Achse. Die Koordinaten I, J und K geben den Kreismittelpunkt in absoluten Koordinaten an. Der Radius wird aus dem Abstand der letzten Position zum Kreismittelpunkt berechnet.
Syntax:
```
G02 G09 [M72] Wf [G90] If Jf       ' XY-Eb.
G02 G09 [M72] Wf [G90] Jf Kf       ' YZ-Eb.
G02 G09 [M72] Wf [G90] Kf If       ' ZX-Eb.
```
Beispiel:
```
G02 G09 W100 I50 J50
G02 G09 M72 W-20 G90 I22.2 J22.2
```

Polarkoordinaten, absoluter Mittelpunkt, relativer Winkel:

Der Winkel W ist der Endwinkel der Kreisfahrt relativ zum Winkel der letzten Position auf dem definierten Kreis. Die Koordinaten I, J und K geben den Kreismittelpunkt in absoluten Koordinaten an. Der Radius wird aus dem Abstand der letzten Position zum Kreismittelpunkt berechnet.

Syntax:

```
G02 G09 M71 Wf [G90] If Jf          ' XY-Eb.
G02 G09 M71 Wf [G90] Jf Kf          ' YZ-Eb.
G02 G09 M71 Wf [G90] Kf If          ' ZX-Eb.
```

Beispiel:

```
G02 G09 M71 W100 I50 J50
G02 G09 M71 W-20 G90 I22.2 J22.2
```

Polarkoordinaten, relativer Mittelpunkt, absoluter Winkel:
Der Winkel W ist der Endwinkel der Kreisfahrt bezüglich der positiven X-Achse. Die Koordinaten I, J und K geben den Kreismittelpunkt relativ zur letzten Position an. Der Radius wird aus dem Abstand der letzten Position zum Kreismittelpunkt berechnet (d.h. aus I und J).

Syntax:

```
G02 G09 [M72] Wf G91 If Jf          ' XY-Eb.
G02 G09 [M72] Wf G91 Jf Kf          ' YZ-Eb.
G02 G09 [M72] Wf G91 Kf If          ' ZX-Eb.
```

Beispiel:

```
G02 G09 W100 G91 I50 J50
G02 G09 M72 W-20 G91 I22.2 J22.2
```

Polarkoordinaten, relativer Mittelpunkt, relativer Winkel:
Der Winkel W ist der Endwinkel der Kreisfahrt relativ zum Winkel der letzten Position auf dem definierten Kreis. Die Koordinaten I, J und K geben den Kreismittelpunkt relativ zur letzten Position an. Der Radius wird aus dem Abstand der letzten Position zum Kreismittelpunkt berechnet.

Syntax:

```
G02 G09 M71 Wf G91 If Jf            ' XY-Eb.
G02 G09 M71 Wf G91 Jf Kf            ' YZ-Eb.
G02 G09 M71 Wf G91 Kf If            ' ZX-Eb.
```

Beispiel:

```
G02 G09 M71 W100 G91 I50 J50
G02 G09 M71 W-20 G91 I22.2 J22.2
```

Siehe auch:
FMAX, VEL, G03

G03

Anweisung

Kreisfahrt gegen den Uhrzeigersinn (positive Winkelrichtung) zu den angegebenen Koordinaten. Die Koordinatenangaben und die Syntax sind die gleichen wie bei der G02-Anweisung.

Siehe auch:
FMAX, VEL, G02

G53

Anweisung

Verschieben und Drehen des Koordinatensystems aufheben.

Syntax:

```
G53
```

Siehe auch:

```
G54, G55, G56
```

G54

Anweisung

Verschieben des Koordinatensystems. Diese Verschiebung ist unabhängig von der mit `SETPOS` eingestellten Verschiebung und wirkt sich nur auf `G-Code`-Anweisungen aus. Die aktuellen Position wird als Position mit den Koordinaten `X`, `Y` und `Z` übernommen.

Syntax:

```
G54 Xf Yf Zf
```

Siehe auch:

```
G53, G55, G56
```

G55

Anweisung

Additives Drehen und Verschieben des Koordinatensystems. Diese Verschiebung und Drehung ist unabhängig von der mit `SETPOS` und `TRAFO` eingestellten Verschiebung und Drehung und wirkt sich nur auf `G-Code`-Anweisungen aus. Das Koordinatensystem wird in der XY-Achse um den relativen Winkel `W` gedreht und um die relativen Koordinaten `X`, `Y` und `Z` verschoben.

Syntax:

```
G55 Xf Yf Zf Wf
```

Siehe auch:

```
G53, G54, G56
```

G56

Anweisung

Absolutes Drehen und Verschieben des Koordinatensystems. Diese Verschiebung und Drehung ist unabhängig von der mit `SETPOS` und `TRAFO` eingestellten Verschiebung und Drehung und wirkt sich nur auf `G-Code`-Anweisungen aus. Das Koordinatensystem wird in der XY-Achse so gedreht, daß die positive X-Achse in Richtung des Winkels `W` zeigt und der Nullpunkt wird so verschoben, daß die aktuellen Koordinaten `X`, `Y` und `Z` lauten.

Syntax:

```
G56 Xf Yf Zf Wf
```

Siehe auch:

```
G53, G54, G55
```

M71

Anweisung

Die Winkelangabe dieser Zeile ist ein relativer Winkel.

Syntax:

```
M71
```

Siehe auch:
G00, G01, G02, G03

M72 *Anweisung*

Die Winkelangabe dieser Zeile ist ein absoluter Winkel.

Syntax:
M72

Siehe auch:
G00, G01, G02, G03

M80 *Anweisung*

Aufhebung aller Spiegelungen.

Syntax:
M80

Siehe auch:
G53, G54, G55, G56, M81, M82, M83, M84

M81 *Anweisung*

Spiegelung an der Y-Achse, d.h. Vertauschung der Vorzeichen von X- und I-Koordinaten.

Syntax:
M81

Siehe auch:
G53, G54, G55, G56, M80, M82, M83, M84

M82 *Anweisung*

Spiegelung an der X-Achse, d.h. Vertauschung der Vorzeichen von Y- und J-Koordinaten.

Syntax:
M82

Siehe auch:
G53, G54, G55, G56, M80, M81, M83, M84

M83 *Anweisung*

Spiegelung an der XY-Ebene, d.h. Vertauschung der Vorzeichen von Z- und K-Koordinaten.

Syntax:
M83

Siehe auch:
G53, G54, G55, G56, M80, M81, M82, M84

M84 *Anweisung*

Vertauschung der Vorzeichen von X- und I-Koordinaten und der Y- und J-Koordinaten.

Syntax:
M84

Siehe auch:
G53, G54, G55, G56, M80, M81, M82, M83

ON CODE

Anweisung

Stellt die Programmzeile `n` ein, zu der bei dem Ereignis `CODE (sel)` gesprungen werden soll. Dieses Ereignis tritt auf, falls ein G-Code mit dem Buchstaben `sel` (`G`, `E` (Zeilenende), `D`, `M`, `T`) eingegeben wurde.

CODE(G)

Dieses Ereignis tritt auf, sobald ein G-Code eingelesen wurde. Falls ein G-Code mit Parametern programmiert werden soll, ist beim Auftreten des G-Code Ereignisses nur eine Variable zu setzen, die am Zeilenende (`CODE(E)`) nach dem Einlesen aller Parameter ausgewertet wird. Anderenfalls wird die G-Code-Routine ausgerufen, bevor die folgenden Parameter eingelesen wurden. Wenn die Parameter `X`, `Y` oder `Z` verwendet werden sollen, ist darauf zu achten, daß die Anweisung `DEFAULTCODE OFF` verwendet wird. Ein G-Code ohne Parameter kann auch direkt ausgeführt werden. Die Nummer, die nach dem G angegeben wurde, kann mit der Funktion `CODE(G)` ausgelesen werden. Das Vorliegen eines neuen G-Code kann mit der Funktion `NEWCODE(G)` überprüft werden.

CODE(E)

Dieses Ereignis tritt auf, sobald das Ende einer Zeile erreicht wurde. Dann können mit Hilfe der `NEWCODE(sel)` Funktion die in dieser Zeile angebenen Parameter überprüft werden.

CODE(D)

Dieses Ereignis tritt auf, sobald ein D-Code eingelesen wurde. Die Auswertung erfolgt wie bei `CODE(G)`.

CODE(M)

Dieses Ereignis tritt auf, sobald ein M-Code eingelesen wurde. Die Auswertung erfolgt wie bei `CODE(G)`.

CODE(T)

Dieses Ereignis tritt auf, sobald ein T-Code eingelesen wurde. Die Auswertung erfolgt wie bei `CODE(G)`.

Syntax:

```
ON CODE(sel) GOTO|GOSUB n
```

Beispiel:

```
..     Initialisierung
53110 ON CODE(G) GOSUB 72000
53120 ON CODE(E) GOSUB 73000
      hier folgt das Hauptprogramm
..
..     Ereignis-Verarbeitung
72000 '----- G-Code ------------------------
72010 uv_sel=CODE(g)
72020 SELECT CASE uv_sel
72030   CASE 40: LED 1 ON: '  direkte Ausf.
```

```
72040   CASE 41: G41flag=1: ' indir. Ausf.
72100 END SELECT
72999 RETURN

73000 '----- EOL ---------------------------
73010 IF G41flag=1 THEN GOTO 74000: END IF
73999 RETURN

74000 '----- G41 ---------------------------
74010 IF NEWCODE(Z) THEN POSA 2,CODE(Z): END IF
74020 RESETCODEFLAGS
74999 RETURN
```

Siehe auch:
ON, TOOLOFF, TOOLON

Vn *Anweisung*

Wartet n Sekunden.

Syntax:
Vn

G-Code Ergänzungen für spezielle Programme:

Diese Codes sind im Auslieferungszustand nicht definiert, sondern werden bei unterschiedlichen Programmen eingesetzt. Ihre Verwendung setzt die Installation des entsprechenden Programmes voraus. In anderen Programmen sind die Codes zunächst funktionslos, können aber wie auch andere Codes beliebig mit eigenen Funktionen programmiert werden. Die Vorgehensweise ist bei der Anweisung `ON CODE` beschrieben.

Brenner-Steuerung:

G10 — ***Anweisung***

Bewirkt, daß zwischen den Kontursegmenten, zwischen denen die Anweisung steht, nicht abgebremst wird. Verwendbar beispielsweise wenn ein Kreissegment tangential in eine Linearfahrt übergeht.

Syntax:

```
G10
```

Beispiel:

```
G00 X0 Y0
G01 Y50
G10
G02 X100 Y50 I50 J50
G10
G01 Y0
```

G40 — ***Anweisung***

Gibt dem Benutzer die Möglichkeit, die Maschinenposition mit Hilfe des Joysticks zu korrigieren.

Syntax:

```
G40
```

G41 — ***Anweisung***

Wartet mit der Programmausführung, bis der Benutzer die OK-Taste gedrückt hat.

Syntax:

```
G41
```

G42 Vn — ***Anweisung***

Wartet `n` Sekunden mit der Programmausführung.

Syntax:

```
G42 Vn
```

G60 — ***Anweisung***

Fährt mit der Z-Achse in die obere Position.

Syntax:

```
G60
```

G61 — ***Anweisung***

Fährt mit der Z-Achse in die untere Position.

Syntax:
`G61`

G62 Za — ***Anweisung***

Fährt mit der Z-Achse auf die angegebene absolute Z-Position.

Syntax:
`G62 Za`

Beispiel:
`G62 Z-20`

G63 Za — ***Anweisung***

Fährt mit der Z-Achse auf die angegebene relative Z-Position.

Syntax:
`G63 Za`

Beispiel:
`G63 Z10`

4. Anhang B: Programmierhinweise für spezielle Hardware

Programmierung der Mikroschrittendstufen:

Die Mikroschrittendstufe in den Steuerungen CO6100, CO6500 und CO6150 müssen vor dem Einschalten der Endstufen zunächst mit dem gewünschten Verlauf der Phasenströme programmiert werden. Üblicherweise verwendet man dazu sinus- und cosinusförmige Stromverläufe, es ist jedoch auch eine Anpassung an bestimmte Motoren möglich, indem ein motorspezifischer Stromverlauf programmiert wird. Dadurch kann die Positioniergenauigkeit erhöht und die Laufruhe verbessert werden.
Zur Programmierung der Ströme werden an den Stromregler zwei `FIX`-Arrays der Länge 256 übergeben, in denen der Stromverlauf eines vollen Zyklus (entspricht 4 Vollschritten) für jeweils eine Phase übergeben wird. Dadurch wird eine Auflösung von bis zu 1/64 Schritt erreicht. Die Stromwerte sind bei den Steuerungen CO6100 und CO6500 zwischen -1 und +1 skaliert zu übergeben, bei CO6150 zwischen 0 und 1.
Für die Programmierung der Phase 1 wird die `SET na,WAVEFORM1,sinarray[0]` Anweisung verwendet, für Phase 2 `SET na,WAVEFORM2,cosarray[0]`.
Die Polarität ist bei CO6100 und CO6500 frei wählbar, so daß z.B. auch 1/32 Schritt Auflösung programmiert werden kann, indem statt eines Zyklus zwei volle Zyklen abgelegt werden.
Bei der Steuerung CO6150 ist die Polarität fest vorgegeben, so daß nur ein sinus- bzw. cosinusförmiger Verlauf programmiert werden kann. Die Polarität ist bei `WAVEFORM1` auf positiv für den Arraybereich von 1 bis 128 und negativ von 129 bis 256 festgelegt (geeignet für Sinus). `WAVEFORM2` ist positiv von 1 bis 64 und 193 bis 256 und negativ von 65 bis 192 (geeignet für Cosinus).
Die Phasenströme ergeben sich aus dem programmierten Wert und dem Maximalstrom der Endstufen, multipliziert mit den durch die Befehle `SET na,IMAX,a` und `SET na,ISTANDBY,a` vorgegebenen Prozentsatz. Der Maximalstrom der Steuerungen CO6100 und CO6500 beträgt 4 A. Der Maximalstrom der Steuerung CO6150 ist mit einem Potentiometer zwischen 1.2 A und 4.8 A einstellbar, 3 A sollten jedoch auf Dauer nicht überschritten werden.

Beispiel für die Programmierung der Phasenströme bei CO6100 und CO6500:

```
MODE 1,SM,STEPDIR(1)
DIM sinwave(256) AS FIX
DIM coswave(256) AS FIX
FOR i = 1 TO 256: sinwave(i)=SIN((i-1)*2*PI/256): NEXT
FOR i = 1 TO 256: coswave(i)=COS((i-1)*2*PI/256): NEXT
SET 1,WAVEFORM1,sinwave(0)
SET 1,WAVEFORM2,coswave(0)
SET 1,ISTANDBY,40
SET 1,IMAX,70
```

Beispiel für die Programmierung der Phasenströme bei CO6150:

```
MODE 1,SM,STEPDIR(1)
```

```
DIM sinwave(256) AS FIX
DIM coswave(256) AS FIX
FOR i = 1 TO 256: sinwave(i)=ABS(SIN((i-1)*2*PI/256)): NEXT
FOR i = 1 TO 256: coswave(i)=ABS(COS((i-1)*2*PI/256)): NEXT
SET 1,WAVEFORM1,sinwave(0)
SET 1,WAVEFORM2,coswave(0)
SET 1,ISTANDBY,40
SET 1,IMAX,70
```

Für weitere Achsen kann dasselbe Array verwendet werden. Für hochpräzise Positionierungen sollten die Werte für `IMAX` und `ISTANDBY` identisch sein, da durch das Absenken des Stromes auch bei gleichbleibendem Verhältnis der Phasenströme eine Bewegung des Schrittmotors erfolgen kann. Dies wird durch Nichtlinearitäten der Drehmomentkennlinie des Schrittmotors verursacht.

Hinweise zu den Mikroschrittendstufen der Steuerung CO6300

Im Gegensatz zu den Endstufen der Steuerungen CO6100, CO6150 und CO6500 ist der Stromverlauf bei den Endstufen der Steuerung CO6300 bereits fest sinus- und cosinusförmig vorgegeben (für andere Funktionen bitte anfragen). Daher ist die Übergabe eines Arrays bei der Initialisierung nicht nötig. Die Endstufen arbeiten mit einer Auflösung von 128 Schritten pro voller Phase, was 1/32 Schritt entspricht.

Beispiel für die Programmierung der Phasenströme bei CO6300:

```
MODE 1,SM,STEPDIR(1)
SET 1,ISTANDBY,40
SET 1,IMAX,70
```

Verwendung eines Digitalpotentiometers als Eingabehilfe:

Die Position des Inkrementalgebers wird in einer Timer-Routine abgefragt und entsprechend der Positionsänderung werden Cursor-Up und Cursor-Down Zeichenfolgen generiert. Die Zeichenfolgen sind entsprechend der üblichen Terminalkonvention: ESC [A für Cursor-Up und ESC [B für Cursor-Down.
Das Keyboard-Device kann dazu zum Schreiben geöffnet werden. Alle geschriebenen Zeichen werden dann wie Tastendrücke weiterverarbeitet und ermöglichen damit z.B. das Scrollen innerhalb einer Menüauswahl.
Im den folgenden Beispielprogrammfragmenten entsprechen 4 Inkremente einem Cursor-Up bzw. Cursor-Down Tastendruck. Bei anderen Übersetzungsverhältnissen ist dieser Wert entsprechend anzupassen.

Initialisierung:

```
 950 OPEN "KEY:" FOR OUTPUT AS #2
 960 wheelscale=4
 970 wheelpos=INC(1)/wheelscale
 980 lastwheelpos=INC(1)/wheelscale
 990 wheelflag=0
1000 ON TIMER(50) GOSUB 20000: TIMER ON
1010 ...
```

Verwendung bei einem Menü:

```
3000 wheelflag=1
3010 menselect=MENU("CONFIG",menlist(0))
3020 wheelflag=0
```

Timerroutine:

```
20000 '----------------- TIMER ----------------------------
20010 IF wheelflag=0 THEN GOTO 20200: END IF
20020 wheelpos=INC(1)/wheelscale
20030 wheeldiff=wheelpos-lastwheelpos
20040 IF wheeldiff<0 THEN GOTO 20120: END IF
20050 wheeldiff=CINT(wheeldiff-0.010)
20060 IF wheeldiff=0 THEN GOTO 20200: END IF
20070 lastwheelpos=wheelpos
20080 FOR i=1 TO wheeldiff
20090   PRINT #2,CHR$(27);CHR$(91);CHR$(65);
20100 NEXT i
20110 GOTO 20200
20120 wheeldiff=CINT(ABS(wheeldiff)-0.010)
20130 IF wheeldiff=0 THEN GOTO 20200: END IF
20140 lastwheelpos=wheelpos
20150 FOR i=1 TO wheeldiff
20160   PRINT #2,CHR$(27);CHR$(91);CHR$(66);
20170 NEXT i
20200 ' weitere Timer-Routinen
20210 ...
20900 RETURN
```

5. Anhang C: Ein- und Ausgänge, Übersicht und Beschaltung

I1 Eingang vom Typ 1
I2 Eingang vom Typ 2

O1 Ausgang vom Typ 1
O2 Ausgang vom Typ 2
O3 Ausgang vom Typ 3

Eingänge

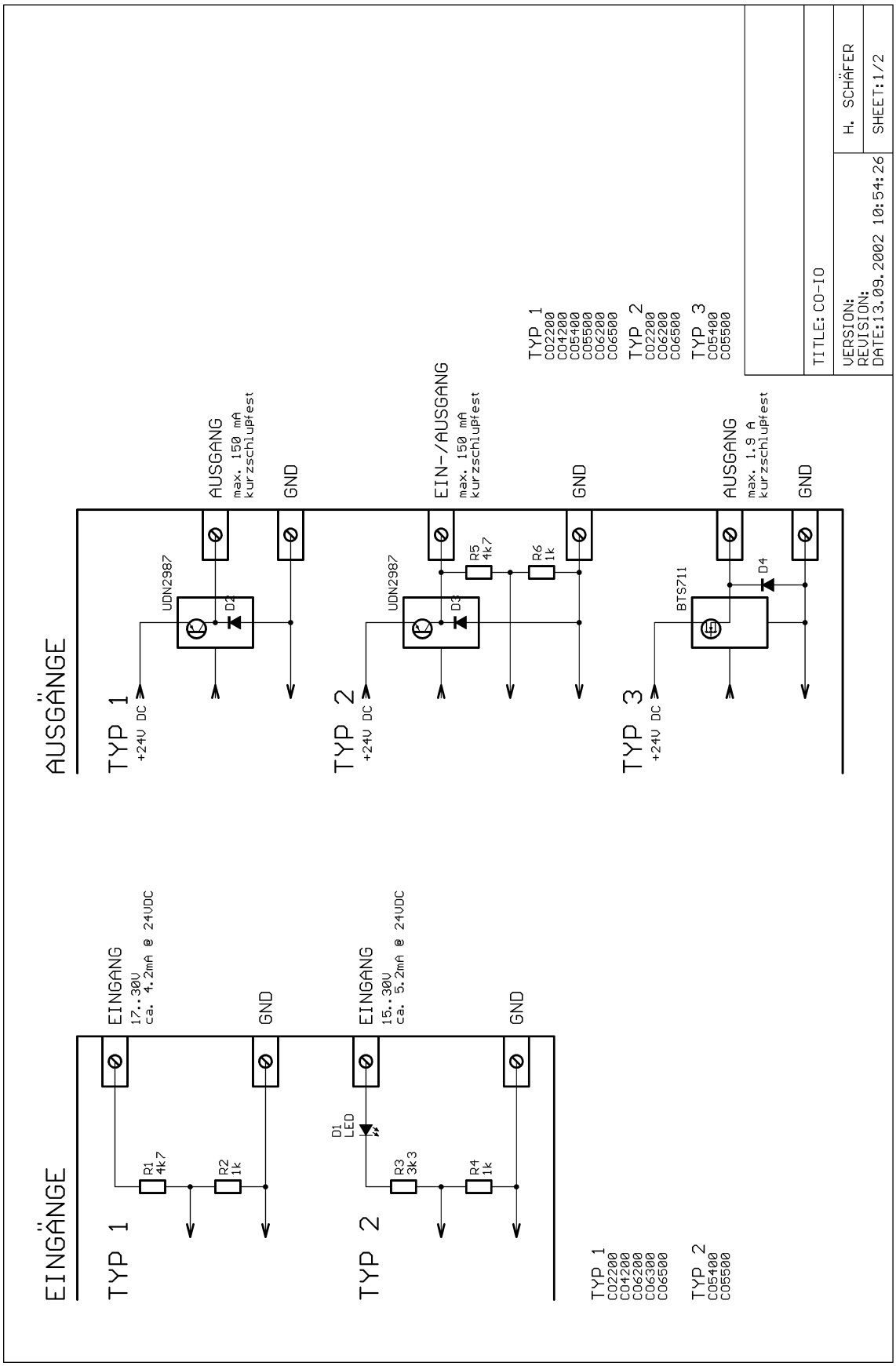
#	CO2200	CO4200 CO4300	CO6100 CO6200	CO6500	CO5400 CO5500
1	IN1 I1	IN1 (1) I1	IO1 O2	IO1 O2	IN1 (1) I2
2	IN2 I1	IN2 (1) I1	IO2 O2	IO2 O2	IN2 (1) I2
3	IN3 I1	IN3 (1) I1	IO3 O2	IO3 O2	IN3 (1) I2
4	IN4 I1	IN4 (1) I1	IO4 O2	IO4 O2	IN4 (1) I2
5	IN5 I1	IN5 (1) I1	IO5 O2	IO5 O2	IN5 (1) I2
6	IN6 I1	IN6 (1) I1	IO6 O2	IO6 O2	IN6 (1) I2
7	IN7 I1	IN7 (1) I1	IO7 O2	IO7 O2	IN7 (1) I2
8	IN8 I1	IN8 (1) I1	IO8 O2	IO8 O2	IN8 (1) I2
9	IO1 O2	IN9 (2) I1	IN9 I1	IN9 I1	REF1 (2) I2
10	IO2 O2	IN10 (2) I1	IN10 I1	IN10 I1	REF2 (2) I2
11	IO3 O2	IN11 (2) I1	IN11 I1	IN11 I1	REF3 (2) I2
12	IO4 O2	IN12 (2) I1	IN12 I1	IN12 I1	REF4 (2) I2
13	IO5 O2	IN13 (2) I1	IN13 I1	IN13 I1	S+ (2) I2
14	IO6 O2	IN14 (2) I1	IN14 I1	IN14 I1	FAULT1 (2)
15	IO7 O2	IN15 (2) I1	IN15 I1	IN15 I1	FAULT2 (2)
16	IO8 O2	IN16 (2) I1	IN16 I1	IN16 I1	EXT (2)
17	PI0	IN17 (3) I1			INDEX1 (3)
18	PI1	IN18 (3) I1			INDEX2 (3)
19	PI2	IN19 (3) I1			INDEX3 (3)
20	PI3	IN20 (3) I1			
21	ERROR1	ERROR1			
22	ERROR2	ERROR2			
23	ERROR3	ERROR3			
24	S+	ERROR4			
25		INDEX1 (4)			
26		INDEX2 (4)			
27		INDEX3 (4)			
28		INDEX4 (4)			
29		-			

#	CO2200	CO4200 CO4300	CO6100 CO6200	CO6500	CO5400 CO5500
30		-			
31		-			
32		-			
33		PI0 (5)			
34		PI1 (5)			
35		PI2 (5)			
36		PI3 (5)			
37		FAULT (5)			
38		S+ (5)			

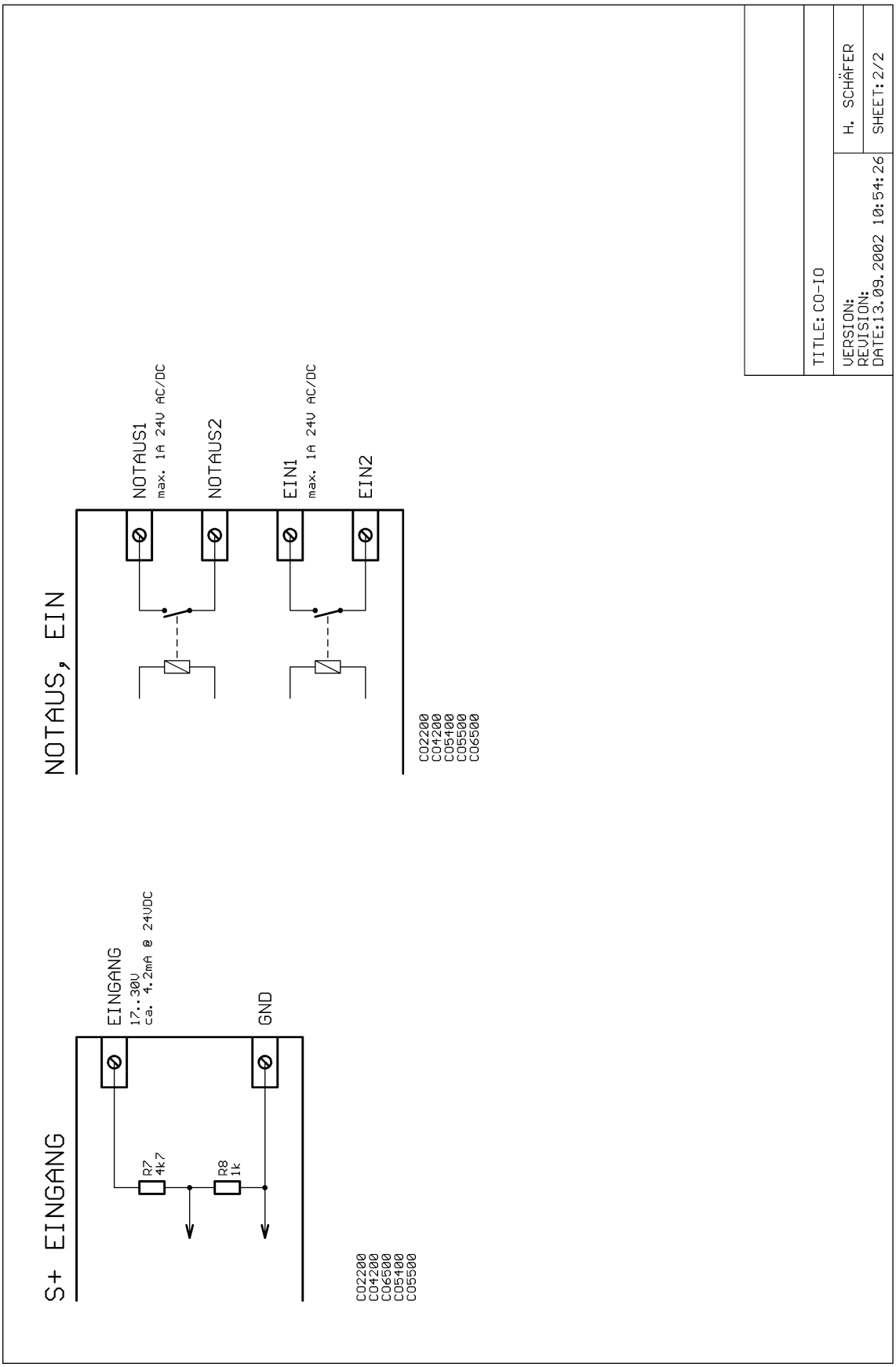
Ausgänge

#	CO2200	CO4200	CO6100 CO6200	CO6500	CO5400 CO5500
1	IO1 O2	OUT1 (1) O1	IO1 O2	IO1 O2	OUT1 (1) O1
2	IO2 O2	OUT2 (1) O1	IO2 O2	IO2 O2	OUT2 (1) O1
3	IO3 O2	OUT3 (1) O1	IO3 O2	IO3 O2	OUT3 (1) O1
4	IO4 O2	OUT4 (1) O1	IO4 O2	IO4 O2	OUT4 (1) O1
5	IO5 O2	OUT5 (1) O1	IO5 O2	IO5 O2	OUT5 (1) O1
6	IO6 O2	OUT6 (1) O1	IO6 O2	IO6 O2	OUT6 (1) O1
7	IO7 O2	OUT7 (1) O1	IO7 O2	IO7 O2	OUT7 (1) O1
8	IO8 O2	OUT8 (1) O1	IO8 O2	IO8 O2	OUT8 (1) O1
9		OUT9 (2) O1			OUT9 (2) O3
10		OUT10 (2) O1			OUT10 (2) O3
11		OUT11 (2) O1			OUT11 (2) O3
12		OUT12 (2) O1			OUT12 (2) O3
13		OUT13 (2) O1			
14		OUT14 (2) O1			
15		OUT15 (2) O1			
16		OUT16 (2) O1			
17					
18					
19					
20					
21					
22					
23					
24					

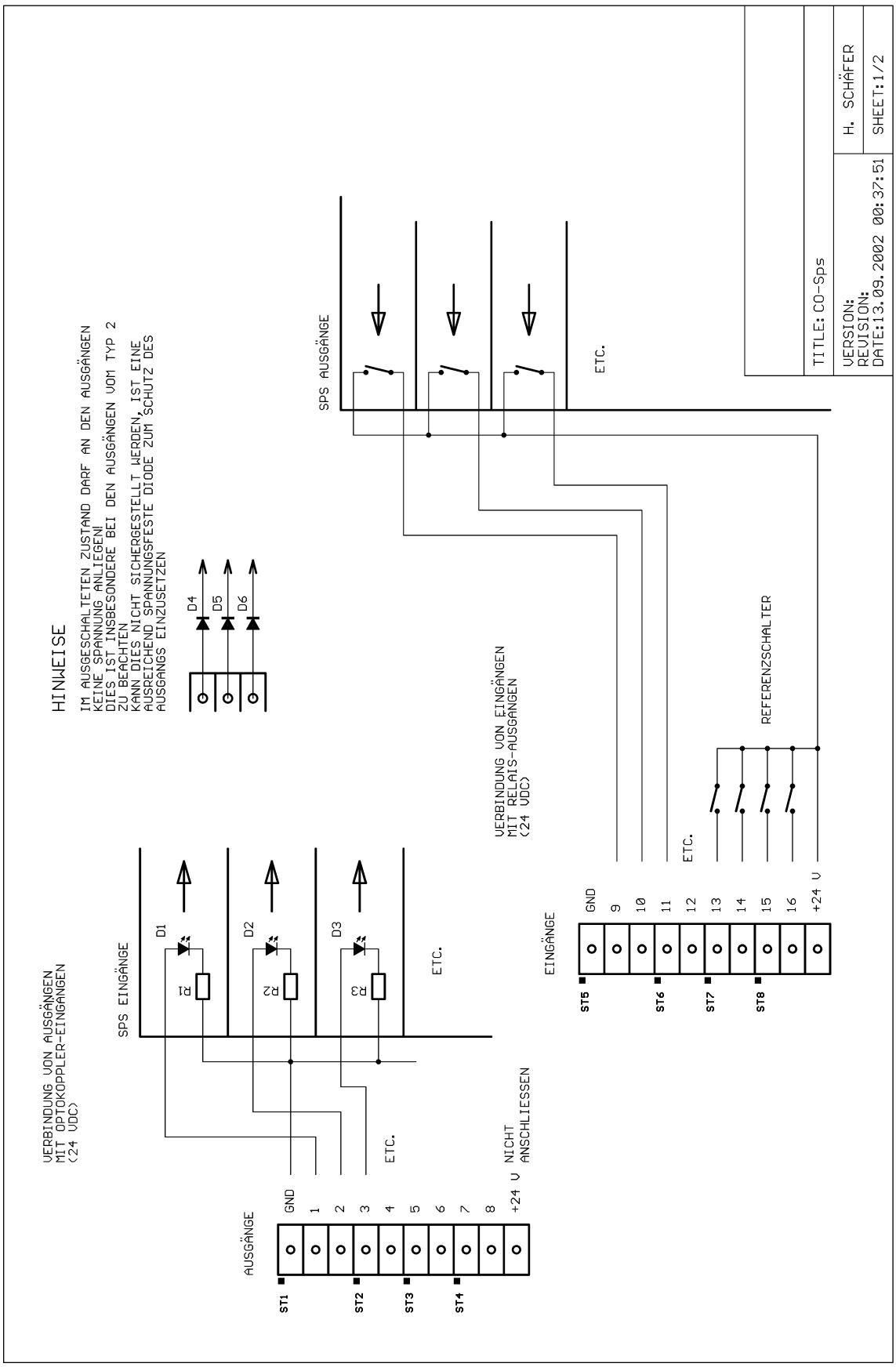
Beschaltung (Ein- und Ausgänge)

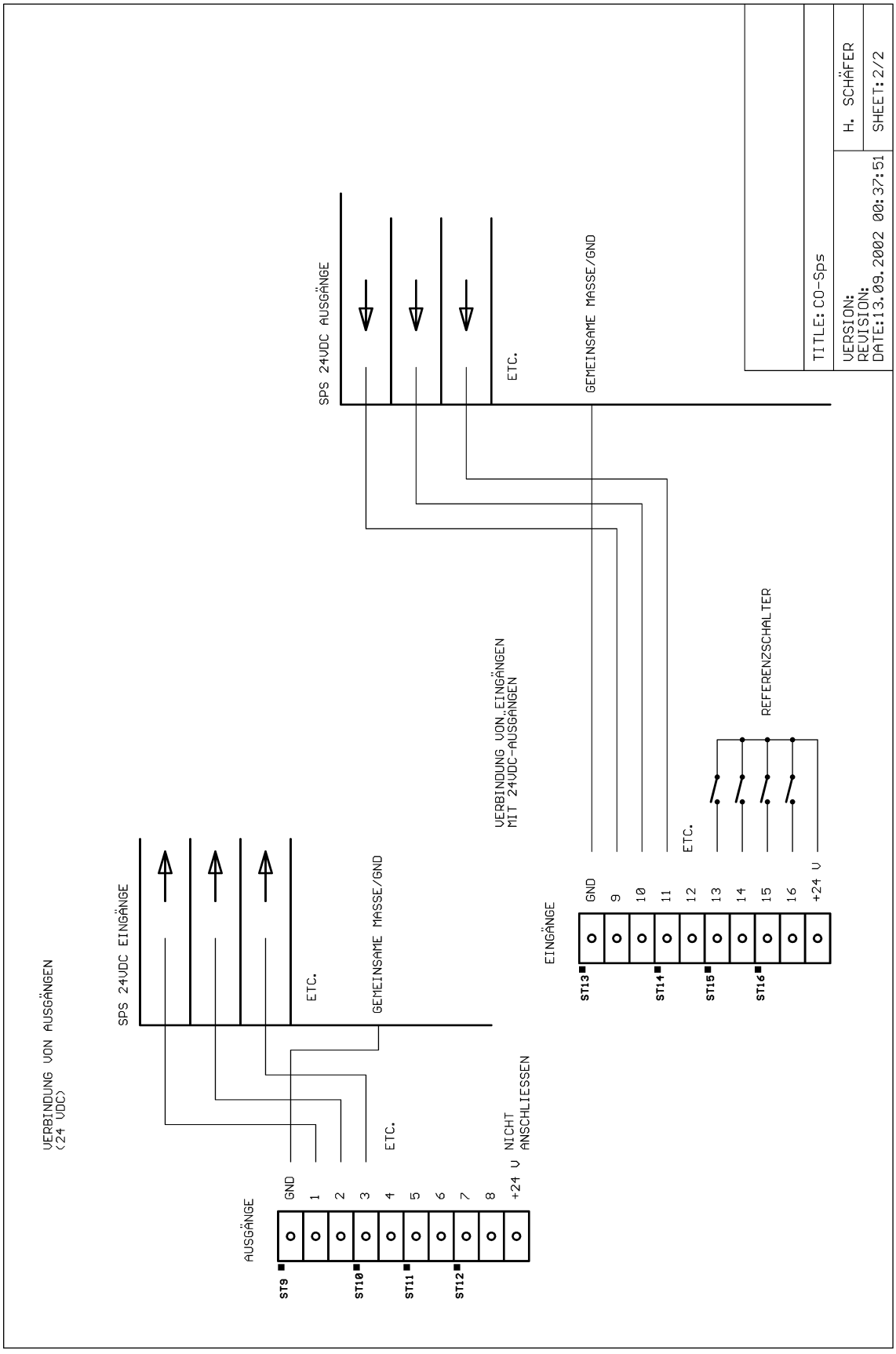


Beschaltung NOTAUS-Funktionen



SPS-Anschluß

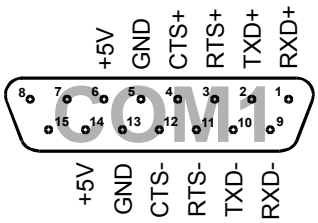


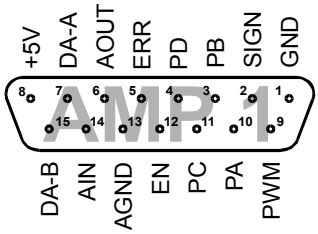

VERBINDUNG VON AUSGÄNGEN
(24 VDC)
SPS 24VDC EINGÄNGE
ETC.
GEMEINSAME MASSE/GND
ETC.
NICHT
ANSCHLIESSEN
+24 V
AUSGÄNGE
GND
1
2
3
4
5
6
7
8
ST9
ST10
ST11
ST12
VERBINDUNG VON EINGÄNGEN
MIT 24VDC-AUSGÄNGEN
SPS 24VDC AUSGÄNGE
ETC.
GEMEINSAME MASSE/GND
REFERENZSCHALTER
EINGÄNGE
GND
9
10
11
12
13
14
15
16
+24 V
ST13
ST14
ST15
ST16
TITLE: CO-Sps
VERSION:
REVISION:
DATE:13.09.2002 00:37:51
H. SCHÄFER
SHEET: 2/2

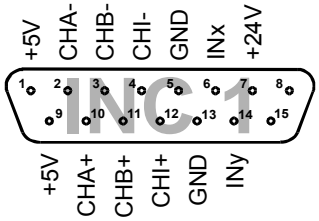
6. Anhang D: ASCII-Code Tabelle

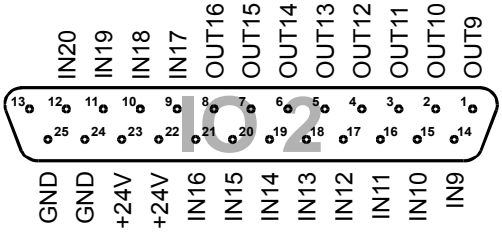
Code	Zeichen	Taste	Funktion	Code	Zeichen	Code	Zeichen	Code	Zeichen
0	Ctrl-A			32	SPACE	64	@	96	`
1	Ctrl-B			33	!	65	A	97	a
2	Ctrl-C			34	“	66	B	98	b
3	Ctrl-D	STOP	BREAK	35	#	67	C	99	c
4	Ctrl-E	HALT		36	$	68	D	100	d
5	Ctrl-F	EDIT		37	%	69	E	101	e
6	Ctrl-G	START		38	&	70	F	102	f
7	Ctrl-H	RUN		39	'	71	G	103	g
8	Ctrl-I	DEL		40	(	72	H	104	h
9	Ctrl-J		TAB	41	)	73	I	105	i
10	Ctrl-J		LF	42	*	74	J	106	j
11	Ctrl-K			43	+	75	K	107	k
12	Ctrl-L		FF	44	,	76	L	108	l
13	Ctrl-M		CR	45	-	77	M	109	m
14	Ctrl-N	MENU		46	.	78	N	110	n
15	Ctrl-O	MOVE		47	/	79	O	111	o
16	Ctrl-P	STEP		48	0	80	P	112	p
17	Ctrl-Q		XON	49	1	81	Q	113	q
18	Ctrl-R	REF		50	2	82	R	114	r
19	Ctrl-S		XOFF	51	3	83	S	115	s
20	Ctrl-T	TEACH		52	4	84	T	116	t
21	Ctrl-U			53	5	85	U	117	u
22	Ctrl-V			54	6	86	V	118	v
23	Ctrl-W			55	7	87	W	119	w
24	Ctrl-X			56	8	88	X	120	x
25	Ctrl-Y			57	9	89	Y	121	y
26	Ctrl-Z			58	:	90	Z	122	z
27		ESC	ESC	59	;	91	[	123	{
28				60	<	92	\	124	|
29				61	=	93	]	125	}
30				62	>	94	^	126	~
31				63	?	95	_	127	

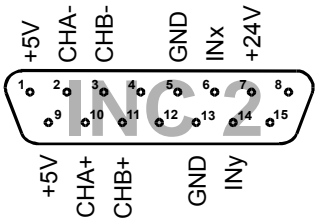
7. Anhang E: Anschlußpläne

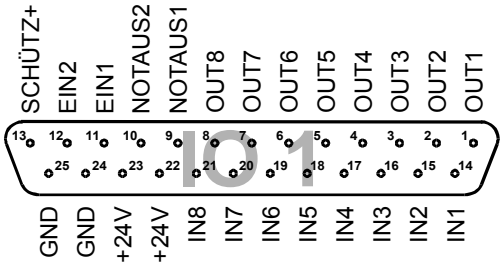


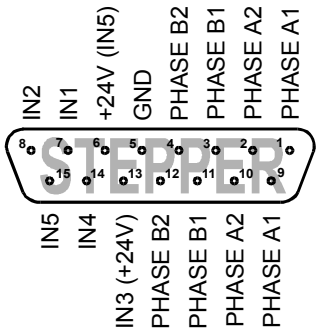


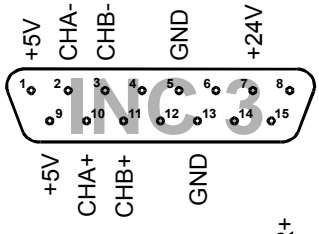


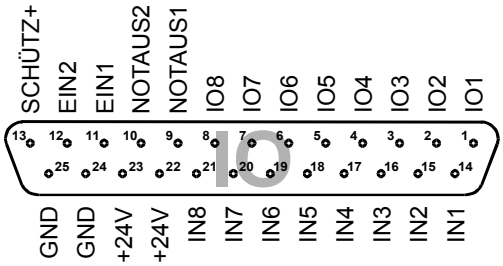

IN20 IN19 IN18 IN17 OUT16 OUT15 OUT14 OUT13 OUT12 OUT11 OUT10 OUT9
IO2
GND GND +24V +24V IN16 IN15 IN14 IN13 IN12 IN11 IN10 IN9

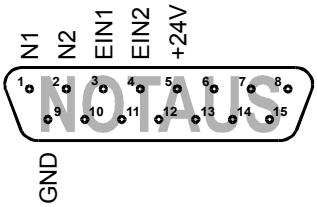


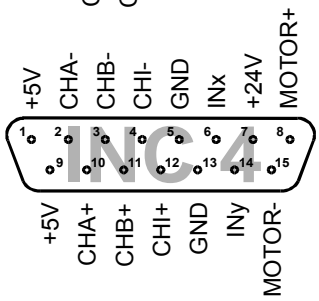


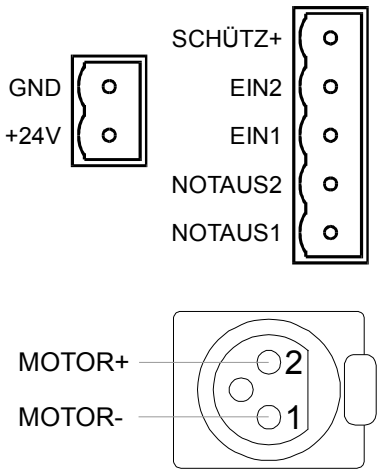


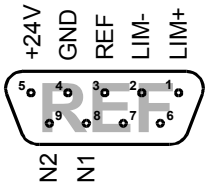


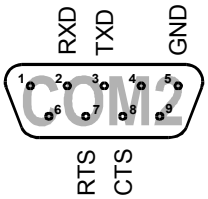


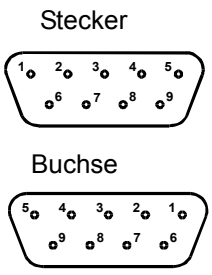


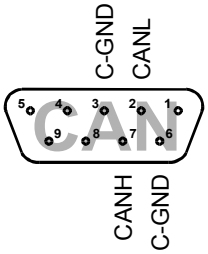




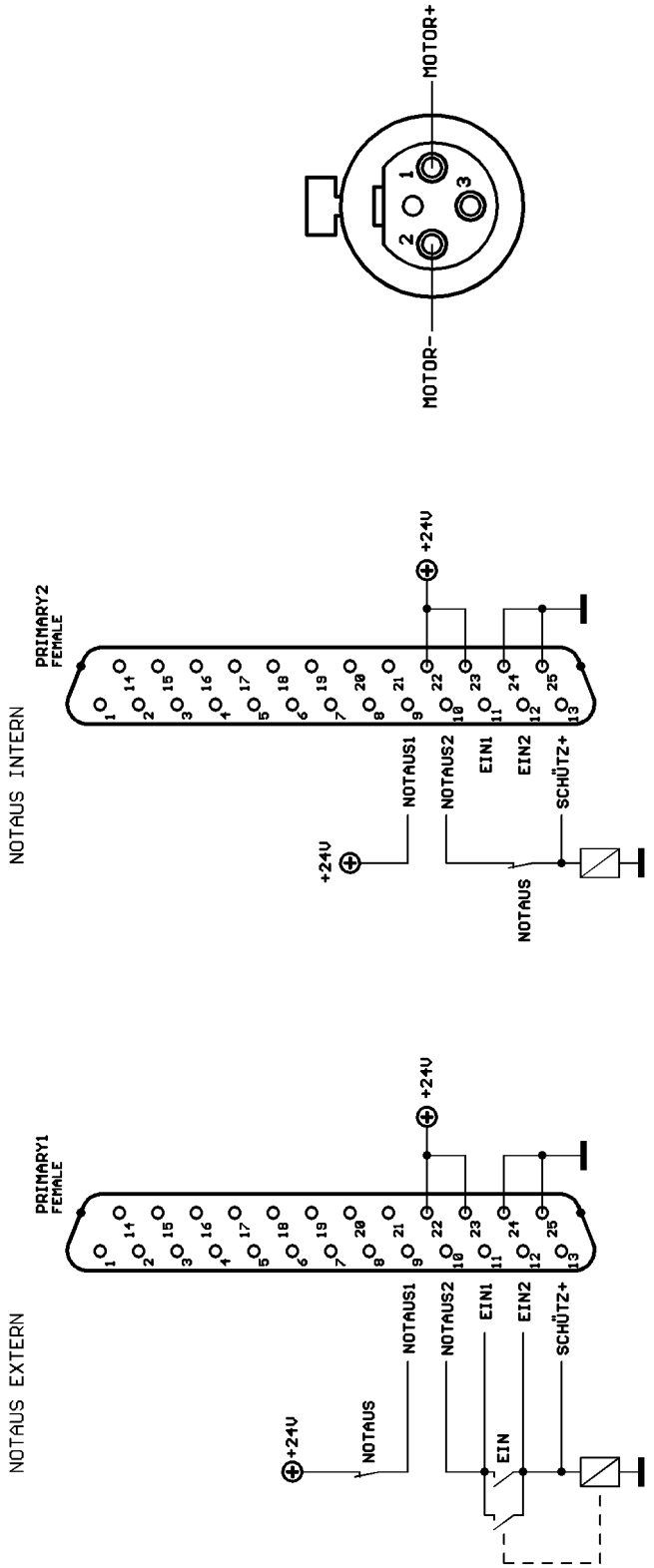








NOTAUSKREIS CO2200, CO4300



8. Anhang F: Beschreibung der Anschlüsse

Name	Funktion
+24V	24 VDC Versorgungsspannung
+5V	+5 VDC zur Versorgung von Inkrementalgebern und Endstufen. Die Inkrementalgeber und die Endstufen dürfen zusammen jeweils 1A verbrauchen
AGND	Analog Masse
AIN	Analoger Eingang +/-10V
AOUT	Analoger Ausgang +/-10V
CANH, CANL	CAN-Bus, externer Abschluß nötig
C-GND	CAN Masse
CHA+, CHA-	Differenzeingänge, 120Ω Abschluß, Inkrementalgeber Kanal A
CHB+, CHB-	Differenzeingänge, 120Ω Abschluß, Inkrementalgeber Kanal B
CHI+, CHI-	Differenzeingänge, 120Ω Abschluß, Inkrementalgeber Index
CTS	RS232 Clear To Send
CTS+, CTS-	RS422 Clear To Send, Differenzeingänge, 120Ω Abschluß
DA-A	Analoger Ausgang 0..5V, Stromeinstellung der Endstufen
DA-B	Analoger Ausgang 0..5V, Stromeinstellung der Endstufen
EIN1, EIN2	Internes EIN-Relais zum Einbinden der START-Taste in einen externen Notauskreis
EN	Enable-Ausgang zum Einschalten der Endstufen, 0V = Aus, +5V = Ein
ERR	Fehler-Eingang, nicht bindende Konvention: 0V = OK, +5V = ERROR
GND	Masse
IN, REF	24 V Eingang, ca. 4 mA Eingangsstrom
IO	24 V Input/Output, Ausgangstrom max. 100 mA (Source) Bei Verwendung als Eingang Pin auf 0 setzen (`PINOUT n,0`)
NOTAUS1, NOTAUS2	Anschluß eines externen Notaustasters. Falls kein externer Notaustaster verwendet wird, müssen diese Kontakte überbrückt werden
OUTn	24 V Ausgang, Ausgangstrom max. 100 mA (Source)
PA	Schrittmotorcontroller Phase A, bzw. Direction
PB	Schrittmotorcontroller Phase B, bzw. Step
PC	Schrittmotorcontroller Phase C, bzw. Standby/Max
PD	Schrittmotorcontroller Phase D
PWM	PWM Ausgang (5V)
RTS	RS232 Ready To Send
RTS+, RTS-	RS422 Ready To Send, Differenzausgänge
RXD	RS232 Empfangsdaten

Name	Funktion
RXD+, RXD-	RS422 Empfangsdaten, Differenzeingänge, 120Ω Abschluß
SCHÜTZ+	Schütz-Testeingang
SIGN	SIGN Ausgang (5V)
TXD	RS232 Sendedaten
TXD+, TXD-	RS422 Sendedaten, Differenzausgänge